Developers Guide

doMail server

Dominanz spol. s r.o.

Contents

List of Figures

1 SMTP

What is SMTP? Simple Mail Transfer Protocol (SMTP) is a quick and easy way to send email from one server to another.

How is SMTP different from other email protocols? The main difference between these protocols is that SMTP is the only protocol for sending or pushing email from one unknown mail server to another.
POP and IMAP are protocols for receiving or pulling mail for the recipient from their own mail server.

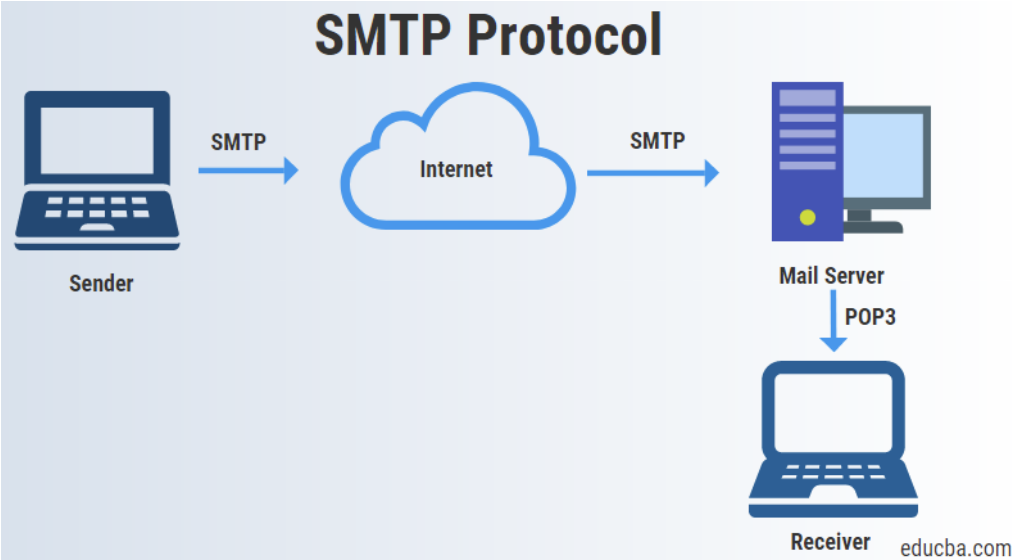

Figure 1: SMTP protocol

By default, SMTP to send email lacks encryption and can be used for sending without any protection in place, leaving emails with an SMTP setup susceptible to man-in-the-middle attacks and eavesdropping from bad actors while messages are in transit. SMTPS uses additional SSL or TLS cryptographic protocols for improved security, and the extra "S" stands for SECURE!
Secure SMTP can be achieved by enabling TLS on your mail server. By enabling TLS, you encrypt the SMTP protocol on the transport layer by wrapping SMTP inside a TLS connection. This effectively secures SMTP and transforms it into SMTPS. Port 587 and 465 are both frequently used for SMTPS traffic. Port 587 is often used to encrypt SMTP messages using STARTTLS, which allows the email client to establish secure connections by requesting that the mail server upgrade the connection through TLS.
Port 465 is used for implicit TLS and can be used to facilitate secure communications for mail services. According to the Internet Engineering Task Force (IETF), this is preferred over using STARTTLS on port 587.
Lastly, port 2525 is sometimes also used. Some residential ISPs will block port 25 to stop users from running their own mail servers. To combat this, enthusiasts and small home businesses use port 2525.
SMTPS plays a key role in email security, but it can't protect against all email-based

threats.

1.1 Send email with SMTP in Python

To send an email using Python and the smtplib library to interact with a domail SMTP server, follow these steps:

- First, ensure you have Python installed on your system and that you have the credentials for your email account. You'll need the SMTP server address (**http:\\{yourUrl}**), port number, and your email login details.
- Start by importing the necessary libraries for creating and sending emails. Use smtplib to handle the connection to the SMTP server and email.mime.multipart and email.mime.text to construct the email message. Use ssl provides functions and classes to use Secure Sockets Layer (SSL) and Transport Layer Security (TLS) to secure communication both server and client side.
- Next, configure the SMTP server settings. Specify the server address and port number. If the server requires TLS, initiate a secure connection using the starttls method. Then, log in to the server using your email username and password.
- Compose your email by setting the sender's address, the recipient's address, the subject line, and the body of the email. Use the MIMEMultipart class to create the email message and attach the body content.
- Once the email is composed, send it using the sendmail of the smtplib server instance.
- Finally, log out from the SMTP server and close the connection using the quit method to ensure a clean exit. This process allows you to send an email programmatically using Python with the domail.

The following code example (1) is lets you send personalized emails:

```python
# Step 1 - Import required packages
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
import smtplib, ssl

# Step 2 - Create message object instance
msg = MIMEMultipart()

# Step 3 - Declare SMTP credentials
password = "XXXXXXXXXX"
username = "XXXXXXXXXX"
smtphost = "yourUrl:port"
context = ssl.create_default_context()

# Step 4 - Create the server connection
server = smtplib.SMTP(smtphost)

server.ehlo()

# Step 5 secure the connection
server.starttls(context=context)

server.ehlo()
```

```python

# Step 6 - Authenticate with the server
server.login(username, password)

# Step 7 - Create message body
message = "Test from Python via AuthSMTP"

# Step 8 - Declare message elements
msg['From'] = "sender@example.com"
msg['To'] = "recipient@example.com"
msg['Subject'] = "Test from Python via AuthSMTP"

# Step 9 - Add the message body to the object instance
msg.attach(MIMEText(message, 'plain'))

# Step 10 - Send the message
server.sendmail(msg['From'], msg['To'], msg.as_string())

# Step 11 - Disconnect
server.quit()
```

Listing 1: Example code sending email with SMTP in Python

1.2 Send email with SMTP in Java

1.2.1 Prerequisites

- Java Development Kit (JDK): Ensure you have the JDK installed on your system.
- JavaMail API: Download the JavaMail API and include it in your project's classpath. You can also use Maven or Gradle to include the necessary dependencies.

1.2.2 Steps to Send an Email

- Import the Necessary Classes:
  - You will need to import classes from the java.util, javax.mail and javax.activation packages.
- Set Up the SMTP Server Properties:
  - Configure the SMTP server settings by creating a Properties object. Set the SMTP server address (yourUrl), port number (e.g. 567), and authentication properties.
  - If your SMTP server requires TLS or SSL, configure the properties accordingly.
- Create a Session:
  - Create a Session object using the configured properties.
  - You will need to provide an Authenticator that contains your email username and password.
- Compose the Email:
  - Create a MimeMessage object using the Session.

  - Set the sender's address, recipient's address, subject line, and the body of the email.
  - Use MimeBodyPart and Multipart if you need to attach files or include different content types.

- Send the Email:
  - Use the Transport class to send the MimeMessage object.
  - The send method will handle the connection to the SMTP server and send the email.

Following (2) is the Send Mail in Java using SMTP with SSL authentication:

```java
import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;

public class Main {

    public static void main(String[] args) {
        // change accordingly
        final String username = "XXXXXXXXXX";

        // change accordingly
        final String password = "XXXXXXXXXX";

        // SMTP host
        final String host = "yourUrl";

        //SMTP port
        final String port = "587";

        // Get system properties
        Properties props = new Properties();

        // enable authentication
        props.put("mail.smtp.auth", "true");

        // enable STARTTLS
        props.put("mail.smtp.starttls.enable", "true");

        // Setup mail server
        props.put("mail.smtp.host", host);

        // TLS Port
        props.put("mail.smtp.port", port);

        // creating Session instance referenced to
        // Authenticator object to pass in
        // Session.getInstance argument
        Session session = Session.getInstance(props,
                new javax.mail.Authenticator() {

                    //override the getPasswordAuthentication method
                    protected PasswordAuthentication
                    getPasswordAuthentication() {

                        return new PasswordAuthentication(username,
                                password);
                    }
```

```
                });

        try {

            // compose the message
            // javax.mail.internet.MimeMessage class is
            // mostly used for abstraction.
            Message message = new MimeMessage(session);

            // header field of the header.
            message.setFrom(new InternetAddress("sender@example.com"));

            message.setRecipients(Message.RecipientType.TO, InternetAddress.parse("
     recipient@example.com"));
            message.setSubject("Test from Java via AuthSMTP");
            message.setText("Test from Java via AuthSMTP");

            //send Message
            Transport.send(message);

        } catch (MessagingException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Listing 2: Example code sending email with SMTP in Java

To send an attachment, replace "message.setText()" with the following code (3):

```
    //create MimeBodyPart object and set your message text
    BodyPart messageBodyPart1 = new MimeBodyPart();
    messageBodyPart1.setText("Test from Java via AuthSMTP");

    //create new MimeBodyPart object and set DataHandler object to this object
    MimeBodyPart messageBodyPart2 = new MimeBodyPart();

    //change accordingly
    String filename = "sample.pdf";

    DataSource source = new FileDataSource(filename);
    messageBodyPart2.setDataHandler(new DataHandler(source));
    messageBodyPart2.setFileName(filename);


    //create Multipart object and add MimeBodyPart objects to this object
    Multipart multipart = new MimeMultipart();
    multipart.addBodyPart(messageBodyPart1);
    multipart.addBodyPart(messageBodyPart2);

    //set the multiplart object to the message object
    message.setContent(multipart);
```

Listing 3: Example code sending attachment with SMTP in Java

1.3 Send email with SMTP in Thunderbird

To use Thunderbird for sending emails via the Domail, follow these steps:

1.3.1 Prerequisites

- Download and Install Thunderbird.

- Have your Domail email account credentials (username and password) and SMTP server details (server address and port).

1.3.2 Steps to Configure Thunderbird

- Open Thunderbird
- Add a New Email Account (New > Existing Mail Account)
- Alternatively, if you are setting up Thunderbird for the first time, you will be prompted to create a new email account.
- Enter your name, email address (e.g., admin@example.com), and password.

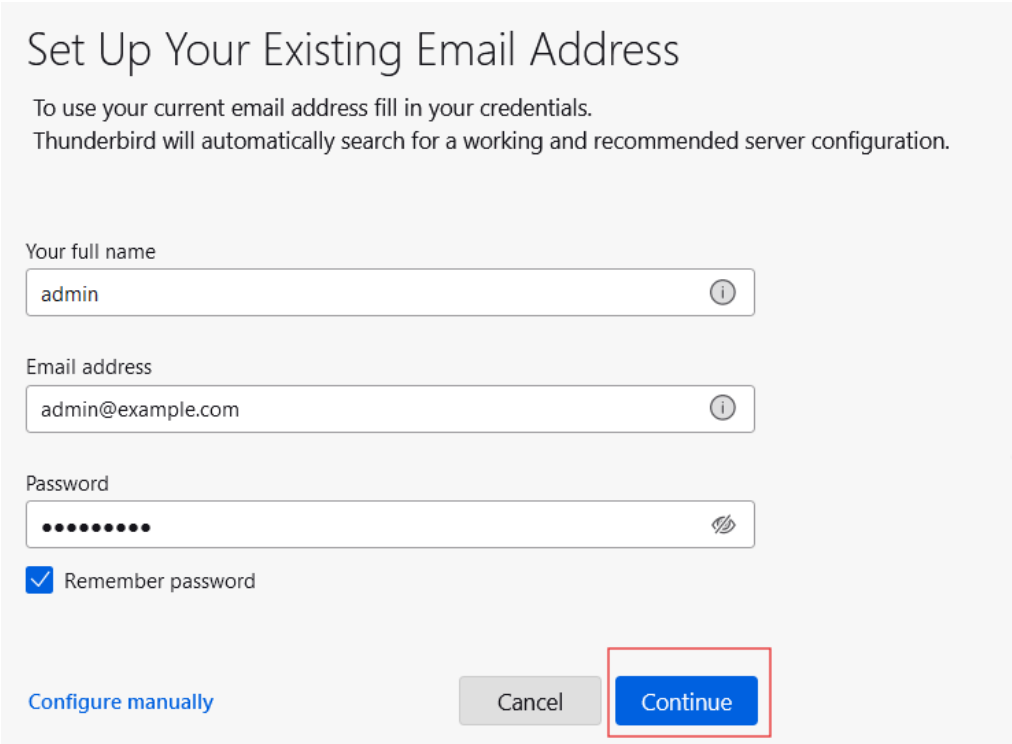

Figure 2: Thunderbird configuration

- Thunderbird will attempt to automatically configure the server settings. If it doesn't find the correct settings, you can manually configure them:
  - Server Name: yourUrl (replace with your actual Domail SMTP server address)
  - Port: 587 (for TLS) or 465 (for SSL)
  - Connection Security: STARTTLS (for port 587) or SSL/TLS (for port 465)
  - Authentication Method: Normal password
  - User Name: admin@example.com (your full email address)
  - Click Re-test to check the settings.

- Click Done to finish setting up your account.

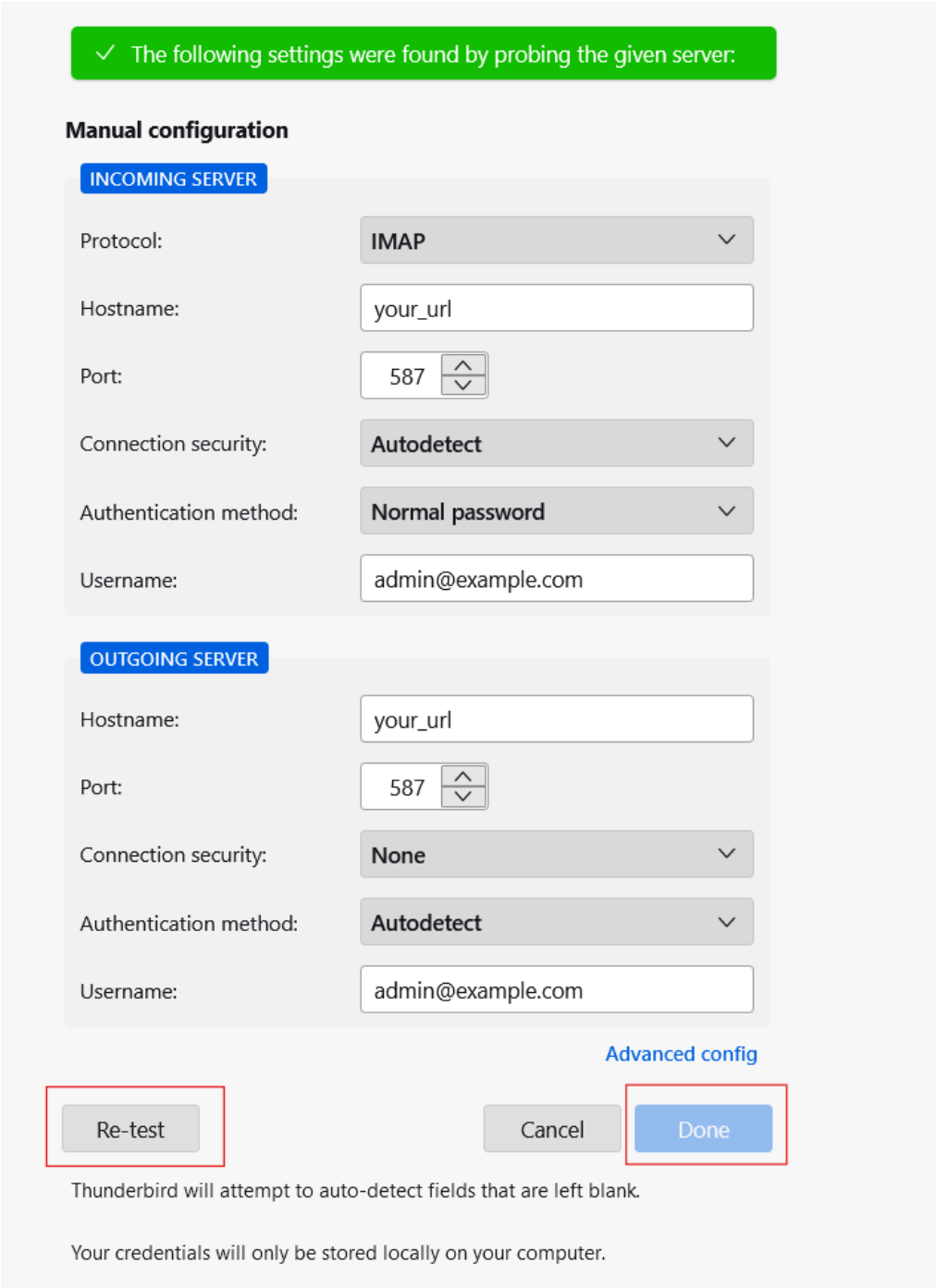

Figure 3: Thunderbird manual configuration

- Go to Account Settings by clicking on your email address in the folder pane and selecting View settings for this account.
- Select Outgoing Server (SMTP) from the list on the left.

- Ensure the correct SMTP server is selected and configured.
- Click Write to compose a new email.
- Enter the recipient's email address, subject, and email body.
- Click Send to send the email using the Domail SMTP server.

2 SOAP

The WSDL file describes the web services provided by the doMail. It outlines the service definitions, operations, messages, bindings, and types used in the communication processes. The WSDL specification provides an XML format for documents for this purpose. WSD can be accessed by calling the endpoint below: **http:\\{yourUrl}/domail**. The definition is written in the file WSDL version 1.1

2.1 Operations

- emailSend – urn:dominanz.sk/domail/emailSend
- emailSendAdv – urn:dominanz.sk/domail/emailSendAdv
- emailSendWithTemplate – urn:dominanz.sk/domail/emailSendWithTemplate
- emailSendWithTemplateAdv – urn:dominanz.sk/domail/emailSendWithTemplateAdv
- communicationGet – urn:dominanz.sk/domail/communicationGet
- communicationGetState – urn:dominanz.sk/domail/communicationGetState

2.1.1 emailSend – urn:dominanz.sk/domail/emailSend

- Endpoint is used to simply send an email via SOAP,
- Body of request is XML object type of 'EmailSendRequestType',
- Response is object type of 'CommunicationResponseType'

In the following code (4)is the definition of the XML object 'EmailSendRequestType':

```
<xsd:element name="emailSendRequestElement" type="EmailSendRequestType">
</xsd:element>
<xsd:complexType name="EmailSendRequestType">
    <xsd:sequence>
        <xsd:element name="to" type="EmailAddressType"
            minOccurs="1" maxOccurs="100">
        </xsd:element>
        <xsd:element name="subject" type="SubjectType"
            minOccurs="0" maxOccurs="1">
        </xsd:element>
        <xsd:element name="htmlBody" type="HtmlTextType"
            minOccurs="0" maxOccurs="1">
        </xsd:element>
        <xsd:element name="textBody" type="PlainTextType"
            minOccurs="0" maxOccurs="1">
        </xsd:element>
        <xsd:element name="attachment"
            type="FileHandlerType" minOccurs="0" maxOccurs="20">
        </xsd:element>
        <xsd:element name="scenario" type="ScenarioType"
            minOccurs="0" maxOccurs="1">
        </xsd:element>
        <xsd:element name="testMode" type="xsd:boolean"
            minOccurs="0" maxOccurs="1">
        </xsd:element>
```

```
        <xsd:element name="statistics" type="StatisticsType" minOccurs="0" maxOccurs="1"
     ></xsd:element>
    </xsd:sequence>
</xsd:complexType>
```

Listing 4: Object type of 'EmailSendRequestType'

- **to** (required field)
  - list of email address of recipients
  - minimum count of recipients is 1 and maximum 100 recipients
  - One recipient of type **EmailAddressType**:
    - **name**
      - Recipient name
      - maximum length is 255 characters
      - string
    - **emailAddress** (required field)
      - Email address of recipient
      - maximum length is 512 characters
      - string
- **subject**
  - subject of email
  - maximum length is 255 characters
  - string
- **htmlBody**
  - html text body of email
  - string
- **textBody**
  - plain text body of email
  - string
- **attachment**
  - array of attachments
  - maximum count of items is 20
  - One item of type **FileHandlerType**:
    - **filename** (required field)
      - File name
      - maximum length is 255 characters
      - string
    - **contentID**
      - Content ID
      - string

        - **mimeType**
            - MimeType (Example: application/text)
            - string
        - **encoding**
            - Encoding (Example: utf-8)
            - string
        - **dataHandler** (required field)
            - Attachment Content
            - Base64 string

- **scenario**
    - the name of the script to be used for this communication
    - pattern: [-0-9a-zA-Z_@#/́]+
    - maximum length is 255 characters
    - string
- **testMode**
    - if the field is set to TRUE, the communication is in test mode
    - default value is FALSE
    - boolean
- **statistics**
    - statistical data that help to classify a given communication
    - **group**
        - pattern: [-0-9a-zA-Z_@#/]+
        - maximum length is 50 characters
        - string
    - **operation**
        - pattern: [-0-9a-zA-Z_@#/]+
        - maximum length is 50 characters
        - string
    - **category**
        - pattern: [-0-9a-zA-Z_@#/]+
        - maximum length is 50 characters
        - string
    - **tag**
        - maximum count of items is 5
            - pattern: [-0-9a-zA-Z_@#/]+
            - maximum length is 50 characters
            - string

In the following code (5)is the definition of the XML object 'CommunicationResponseType':

```
<xsd:element name="communicationResponse" type="CommunicationResponseType">
</xsd:element>
<xsd:complexType name="CommunicationResponseType">
    <xsd:sequence>
        <xsd:element name="id" type="p:long" maxOccurs="1" minOccurs="0"></xsd:element>
    </xsd:sequence>
</xsd:complexType>
```

Listing 5: Object type of 'CommunicationResponseType'

- **id**
  - Id of send communication in the system doMail,
  - It has to be bigger than 0

2.1.2 emailSendAdv – urn:dominanz.sk/domail/emailSendAdv

- Send email (advanced),
- Body of request is XML object type of 'EmailSendAdvRequestType',
- Response is object type of 'CommunicationId'

In the following code (6)is the definition of the XML object 'EmailSendAdvRequestType'. This object type is inherited from object type of 'EmailSendRequestType' (4).

```
    <xsd:element name="emailSendAdvRequest" type="EmailSendAdvRequestType">
    </xsd:element>
    <xsd:complexType name="EmailSendAdvRequestType">
        <xsd:sequence>
            <xsd:element name="emailSendRequest"
                type="EmailSendRequestType">
            </xsd:element>
            <xsd:element name="sysId" type="Std50RegexType"
                minOccurs="1" maxOccurs="1">
            </xsd:element>
            <xsd:element name="extId" type="Std128RegexType"
                minOccurs="1" maxOccurs="1">
            </xsd:element>
            <xsd:element name="advreqdata" type="AdvReqType" minOccurs="0" maxOccurs="1"
     ></xsd:element>
            <xsd:element name="additionalContent"
                type="AdditionalContentType" minOccurs="0"
                maxOccurs="unbounded">
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
```

Listing 6: Object type of 'EmailSendAdvRequestType'

New fields are:

- **sysId** (required field)
  - the unique id/name of the external system wich use this the system doMail
  - pattern: [-0-9a-zA-Z_@#/]+

  - maximum length is 50 characters

- **extId** (required field)
  - the unique id in external system wich use this the system doMail.
  - the doMail checks if duplicity is enabled/disabled. If 'duplicity' is disabled so it will not send more emails with the same 'extId'
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 128 characters

- **advreqdata** - object type is 'AdvReqType', subfields are:
  - **priority**
    * if we want to send a high priority email, we set the value to TRUE
    * boolean
    * value is true/false
  - **duplicity**
    * if we want to send more emails with the same extId, we have to set the value to TRUE
    * boolean
    * value is true/false
  - **sendTime**
    * If it is set, the email will be sent at that time
    * string (date-time)

- **additionalContent** - array of objects type 'AdditionalContentType'. Subfields of object type 'AdditionalContentType' are:
  - **key**
    * Key is unique name of item
    * pattern: [-0-9a-zA-Z_@#/]+
    * maximum length is 50 characters
  - **value**
    * Value of item
    * Base64 string

2.1.3 emailSendWithTemplate – urn:dominanz.sk/domail/emailSendWithTemplate

- Endpoint is used to simply send an email with template via SOAP,
- Body of request is XML object type of 'EmailSendWithTemplateRequestType',
- Response is object type of 'EmailSendWithTemplateResponseType'

In the following code (7)is the definition of the XML object 'EmailSendWithTemplateRequestType':

```
<xsd:element name="emailSendWithTemplateRequest" type="
 EmailSendWithTemplateRequestType">
</xsd:element>
<xsd:complexType name="EmailSendWithTemplateRequestType">
    <xsd:sequence>
        <xsd:element name="scenario" type="ScenarioType"
            minOccurs="1" maxOccurs="1">
        </xsd:element>
        <xsd:element name="statistics" type="StatisticsType"
            minOccurs="0" maxOccurs="1">
        </xsd:element>
        <xsd:element name="testMode" type="xsd:boolean"
            minOccurs="0" maxOccurs="1">
        </xsd:element>
        <xsd:element name="fillParams"
            type="ParametersTemplateType" minOccurs="1"
            maxOccurs="unbounded">
        </xsd:element>
        <xsd:element name="attachments" type="FileHandlerType" minOccurs="0"
 maxOccurs="20"></xsd:element>
    </xsd:sequence>
</xsd:complexType>
```

Listing 7: Object type of ’EmailSendWithTemplateRequestType’

Properties of object type ’EmailSendWithTemplateRequestType’:

- **scenario** (required field)
  - the name of the process to be used for this communication
  - pattern: [-0-9a-zA-Z_@#/́]+
  - maximum length is 255 characters
  - string
- **statistics**
  - statistical data that help to classify a given communication
  - **group**
    - pattern: [-0-9a-zA-Z_@#/]+
    - maximum length is 50 characters
    - string
  - **operation**
    - pattern: [-0-9a-zA-Z_@#/]+
    - maximum length is 50 characters
    - string
  - **category**
    - pattern: [-0-9a-zA-Z_@#/]+
    - maximum length is 50 characters
    - string
  - **tag**
    - maximum count of items is 5
      - pattern: [-0-9a-zA-Z_@#/]+

        - maximum length is 50 characters
        - string

- **testMode**
  - if the field is set to TRUE, the communication will be to send in test mode
  - default value is FALSE
- **fillParams** (required field)
  - list of parameters to be used in the template
  - array of objects type of 'ParametersTemplateType'
  - **key** (required field)
    - string
    - pattern: [-0-9a-zA-Z_@#/]+
  - **value** (required field)
    - string
    - maximum length is 1024 characters
- **attachments**
  - array of attachments
  - maximum count of items is 20
  - One item of type **FileHandlerType**:
    - **filename** (required field)
      - File name
      - maximum length is 255 characters
      - string
    - **contentID**
      - Content ID
      - string
    - **mimeType**
      - MimeType (Example: application/text)
      - string
    - **encoding**
      - Encoding (Example: utf-8)
      - string
    - **dataHandler** (required field)
      - Attachment Content
      - Base64 string

In the following code (8) is the definition of the XML object 'EmailSendWithTemplateResponseType':

```
<xsd:element name="emailSendWithTemplateResponse" type="
     EmailSendWithTemplateResponseType">
</xsd:element>
<xsd:complexType name="EmailSendWithTemplateResponseType">
    <xsd:sequence>
        <xsd:element name="metaCommunication"
            type="p:long" minOccurs="1" maxOccurs="1">
        </xsd:element>
        <xsd:element name="subCommunication"
            type="p:long" minOccurs="0" maxOccurs="unbounded">
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
```

Listing 8: Object type of 'EmailSendWithTemplateResponseType'

Response XML object 'EmailSendWithTemplateResponseType':

- **metaCommunication**
  - id of the main meta communication,
  - It has to be bigger than 0
- **subCommunication** - array of type integer 64bit (long)
  - List of subcommunication IDs that were created based on the defined template

2.1.4 emailSendWithTemplateAdv – urn:dominanz.sk/domail/emailSendWithTemplateAdv

- Send of email with template (Advanced)
- Body of request is XML object type of 'emailSendWithTemplateAdvRequestType',
- Response is object type of 'emailSendWithTemplateResponseType'

In the following code (9)is the definition of the XML object 'emailSendWithTemplateAdvRequestType'.

```
    <xsd:element name="emailSendWithTemplateAdvRequest" type="
     EmailSendWithTemplateAdvRequestType">
    </xsd:element>
    <xsd:complexType
        name="EmailSendWithTemplateAdvRequestType">
        <xsd:sequence>
            <xsd:element name="scenario" type="ScenarioType"
                minOccurs="1" maxOccurs="1">
            </xsd:element>
            <xsd:element name="sysId" type="Std50RegexType"
                minOccurs="1" maxOccurs="1">
            </xsd:element>
            <xsd:element name="extId" type="Std128RegexType"
                minOccurs="1" maxOccurs="1">
            </xsd:element>
            <xsd:element name="statistics" type="StatisticsType"
                minOccurs="0" maxOccurs="1">
            </xsd:element>
            <xsd:element name="testMode" type="xsd:boolean"
```

```
                minOccurs="0" maxOccurs="1">
            </xsd:element>
            <xsd:element name="advReqData" type="AdvReqType"
                minOccurs="0" maxOccurs="1">
            </xsd:element>
            <xsd:element name="attachments"
                type="FileHandlerType" minOccurs="0"
                maxOccurs="unbounded">
            </xsd:element>
            <xsd:element name="templateDataJSON"
                type="templateDataJSONType" minOccurs="0"
                maxOccurs="1">
            </xsd:element>
            <xsd:element name="templateDataCSV" type="templateDataCSVType" minOccurs="0"
    maxOccurs="1"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>
```

Listing 9: Object type of ’emailSendWithTemplateAdvRequestType’

- **scenario** (required field)
  - the name of the process to be used for this communication
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 255 characters
- **sysId** (required field)
  - the unique id/name of the external system wich use this the system doMail
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 50 characters
- **extId** (required field)
  - the unique id in external system wich use this the system doMail.
  - the doMail checks if duplicity is enabled/disabled. If ’duplicity’ is disabled so it will not send more emails with the same ’extId’
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 128 characters
- **statistics** - JSON object type of ’StatisticsType’
  - statistical data that help to classify a given communication
  - **group**
    - pattern: [-0-9a-zA-Z_@#/]+
    - maximum length is 50 characters
  - **operation**
    - pattern: [-0-9a-zA-Z_@#/]+
    - maximum length is 50 characters
  - **category**

        * pattern: [-0-9a-zA-Z_@#/]+
        * maximum length is 50 characters
    - **tag**
        * maximum count of items is 5
        * pattern: [-0-9a-zA-Z_@#/]+

- **testMode**
    - if the field is set to TRUE, the communication will be to send in test mode
    - default value is FALSE
    - boolean
- **advReqData** - object type is 'AdvReqType', subfields are:
    - **priority**
        * if we want to send a high priority email, we set the value to TRUE
        * boolean
        * value is true/false
    - **duplicity**
        * if we want to send more emails with the same extId, we have to set the value to TRUE
        * boolean
        * value is true/false
    - **sendTime**
        * If it is set, the email will be sent at that time
        * string (date-time)
        * value have to be time greater than now
- **attachments**
    - array of attachments
    - maximum count of items is 20
    - One item of type **fileHandlerType**:
        * **filename** (required field)
            · file name
            · maximum length is 255 characters
        * **contentID**
            · Content ID
            · string
        * **mimeType**
            · MimeType (Example: application/text)
            · string
        * **encoding**
            · Encoding (Example: utf-8)
            · string

        * **dataHandler** (required field)
            · Base64 string

- **templateDataJSON**
    - array of any objects
    - Base64 string
- **templateDataCSV**
    - object of type **fileHandlerType** - same as was in attachments:
        * **filename** (required field)
            · file name
            · maximum length is 255 characters
        * **contentID**
            · Content ID
            · string
        * **mimeType**
            · MimeType (Example: application/text)
            · string
        * **encoding**
            · Encoding (Example: utf-8)
            · string
        * **dataHandler** (required field)
            · Base64 string

2.1.5 communicationGet – urn:dominanz.sk/domail/communicationGet

- Get details of given communication request by Id or extID
- Body of request is XML object type of 'CommunicationGetRequestType',
- Response is object type of 'CommunicationGetResponseType'

In the following code (10)is the definition of the XML object 'CommunicationGetRequestType'.

```
<xsd:element name="communicationGetRequest" type="CommunicationGetRequestType">
</xsd:element>
<xsd:complexType name="CommunicationGetRequestType">
    <xsd:sequence>
        <xsd:element name="id" type="p:long" maxOccurs="1" minOccurs="0"></xsd:element>
        <xsd:element name="extID" type="Std128RegexType" minOccurs="0" maxOccurs="1"></xsd:element>
    </xsd:sequence>
</xsd:complexType>
```

Listing 10: Object type of 'CommunicationGetRequestType'

Object of type 'CommunicationGetRequestType' has fields:

- **id**

  - main id of communication

- **extId**
  - the unique id in external system wich use this the system doMail.
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 128 characters

In the following code (11) is the definition of the XML object 'CommunicationGetResponseType':

```
<xsd:element name="communicationGetResponse" type="CommunicationGetResponseType">
</xsd:element>
<xsd:complexType name="CommunicationGetResponseType">
    <xsd:sequence>
        <xsd:element name="detail"
            type="CommunicationDetailType" minOccurs="0" maxOccurs="unbounded">
        </xsd:element>

    </xsd:sequence>
</xsd:complexType>
```

Listing 11: Object type of 'CommunicationGetResponseType'

Response XML object 'CommunicationGetResponseType':

- **detail**
  - object of type **CommunicationDetailType**
    - **id**
      - main id of communication
    - **sysId**
      - the unique id/name of the external system wich use this the system doMail
    - **extId**
      - the unique id in external system wich use this the system doMail.
    - **lastState** - object type is 'StateType', subfields are:
      - **processingState**
      - **deliveryState**
    - **scenario**
      - the name of the scenario that was used for this communication
    - **statistics**
      - statistical data that help to classify a given communication
    - **testMode**
      - the communication was sent in test mode
    - **rcvTime**
    - **advancedReqData**
    - **runs** - object type is 'CommunicationProcDataType', subfields are:
      - **runId**
      - **procTime**
      - **scenarioName**
      - **state** - object type is 'StateType'
      - **sentEmails** - object type is 'SendCommunicationType'

2.1.6 communicationGetState – urn:dominanz.sk/domail/communicationGetState

- Get state of given communication request by Id or extID
- Body of request is XML object type of 'CommunicationGetRequestType',
- Response is object type of 'CommunicationGetStateResponseType'

In the following code (12)is the definition of the XML object 'CommunicationGetRequestType'.

```
<xsd:element name="communicationGetRequest" type="CommunicationGetRequestType">
</xsd:element>
<xsd:complexType name="CommunicationGetRequestType">
    <xsd:sequence>
        <xsd:element name="id" type="p:long" maxOccurs="1" minOccurs="0"></xsd:element>
        <xsd:element name="extID" type="Std128RegexType" minOccurs="0" maxOccurs="1"></xsd:element>
    </xsd:sequence>
</xsd:complexType>
```

Listing 12: Object type of 'CommunicationGetStateRequestType'

Object of type 'CommunicationGetRequestType' has fields:

- **id**
  - main id of communication
- **extId**
  - the unique id in external system wich use this the system doMail.
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 128 characters

In the following code (13) is the definition of the XML object 'CommunicationGetStateResponseType':

```
<xsd:element name="communicationGetStateResponse" type="CommunicationGetStateResponseType">
</xsd:element>
<xsd:complexType name="CommunicationGetStateResponseType">
    <xsd:sequence>
        <xsd:element name="state"
            type="CommunicationStateType" minOccurs="0" maxOccurs="unbounded">
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
```

Listing 13: Object type of 'CommunicationGetStateResponseType'

Response XML object 'CommunicationGetStateResponseType':

- **state** - object type is 'CommunicationStateType', subfields are:
  - **id**
  - **lastState** - object type is 'StateType'
  - **runs** - object type is 'StateRunType'

3 REST API

The definition of the REST API is written in the file YAML in OpenAPI version 3.0.2, which can be downloaded from the doMail website and viewed in Swagger: **http://{domail_address}/swg/swagger-ui.html**

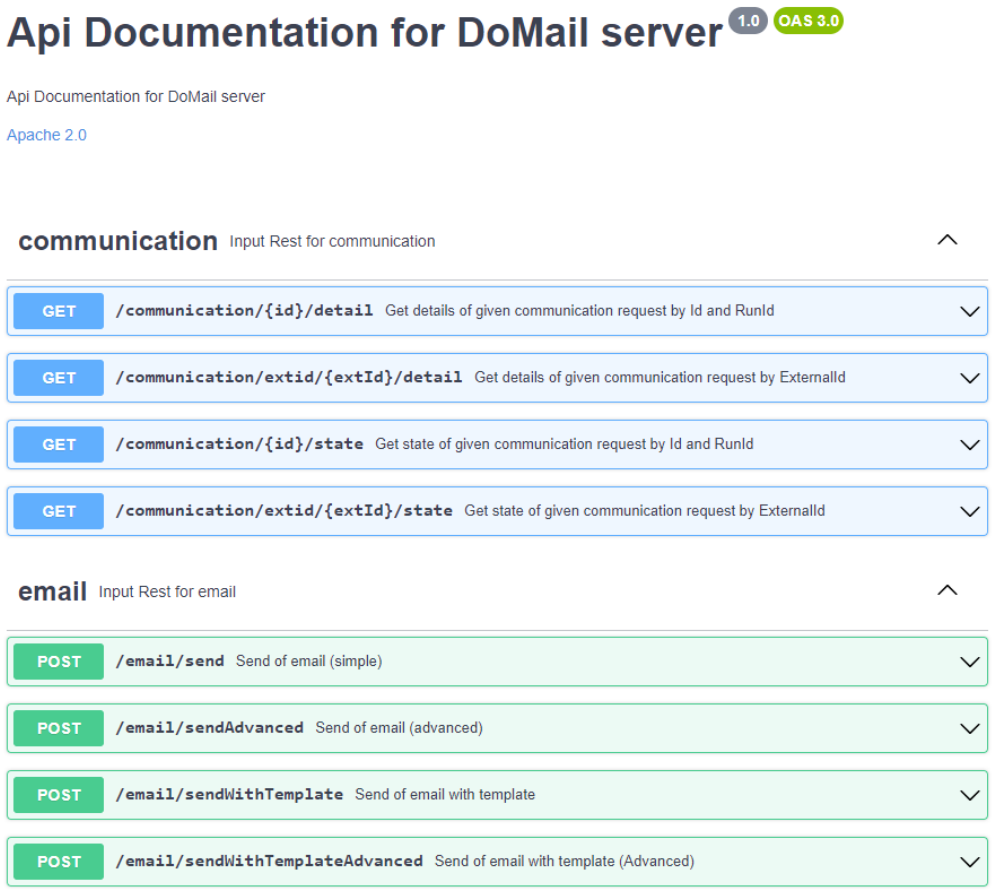

Figure 4: REST API Documentation for DoMail server

3.1 Definition calls

3.1.1 Communication

The following endpoints are available in communication:

- GET /communication/{id}/detail
- GET /communication/extid/{extId}/detail
- GET /communication/{id}/state
- GET /communication/extid/{extId}/detail

3.1.1.1 /communication/{id}/detail

- HTTP GET request
- Get details of given communication request by Id
- The input parameter is the id. This is the return value when a request is successfully sent to the doMail system.
- Response is JSON object type of 'CommunicationDetail'. The model of the 'CommunicationDetail' object is shown in the following 2 figures 5 and 6.

```
communicationDetail ⌄ {
    id*                     integer($int64)
    rcvTime                 string($date-time)
    sysId                   std50Regex  string
                            pattern: [-0-9a-zA-Z_@#/]+
                            maxLength: 50
    extId                   std128Regex  string
                            pattern: [-0-9a-zA-Z_@#/]+
                            maxLength: 128
    lastProcessingState     processingStateType  string
                            Enum:
                             ⌄ [ RECEIVED, COMPLETE, FAILED, DUPLICATE, PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED, DISPATCH_ERROR, CANCELLED ]
    lastDeliveryState       deliveryStateType  string
                            Enum:
                             ⌄ [ DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS ]
    scenario                std255Regex  string
                            pattern: [-0-9a-zA-Z_@#/]+
                            maxLength: 255
    testmode                boolean
                            default: false
    statistics              statisticsType ⌄ {
        group                   std50Regex  string
                                pattern: [-0-9a-zA-Z_@#/]+
                                maxLength: 50
        operation               std50Regex  string
                                pattern: [-0-9a-zA-Z_@#/]+
                                maxLength: 50
        category                std50RegexOpt  string
                                maxLength: 50
                                pattern: [-0-9a-zA-Z_@#/]*
        tags                     ⌄ [
                                maxItems: 5
                                std50Regex  string
                                pattern: [-0-9a-zA-Z_@#/]+
                                maxLength: 50]
                            }
```

Figure 5: Model of object CommunicationDetail 1

```
advancedRequestData   advancedRequestType ⌄ {
                        priority            boolean
                        duplicity           boolean
                        sendTime            string($date-time)

                      }
runs                   ⌄ [communicationProcDataType ⌄ {
                        runId               integer
                        procTime            string($date-time)
                        scenarioName        string255  string
                                            maxLength: 255
                        processingState     processingStateType  string
                                            Enum:
                                             ⌄ [ RECEIVED, COMPLETE, FAILED, DUPLICATE, PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED, DISPATCH_ERROR,
                                            CANCELLED ]
                        deliveryState       deliveryStateType  string
                                            Enum:
                                             ⌄ [ DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS ]
                        sentEmails           ⌄ [sendCommunicationType ⌄ {
                          addressNumber*      integer
                          addressName         addressName  string
                                              maxLength: 200
                          emailAddress        emailAddress  string
                                              maxLength: 512
                                              pattern: [^@]+@[^.]+..+
                          processingState     processingStateType  string
                                              Enum:
                                               ⌄ [ RECEIVED, COMPLETE, FAILED, DUPLICATE, PROCESSING, BLACKLISTED, DISPATCHING,
                                              DISPATCHED, DISPATCH_ERROR, CANCELLED ]
                          deliveryState       deliveryStateType  string
                                              Enum:
                                               ⌄ [ DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED,
                                              DELIVERY_AMBIGUOUS ]
                          messageId           string80  string
                                              maxLength: 80
                          sendTime            string($date-time)

                         }]

                       }]
}
```

Figure 6: Model of object CommunicationDetail 2

Object of type ’CommunicationDetail’ has fields:

- **id** (required field)
  - main id of communication
- **rcvTime**
  - time of receipt of the communication
- **sysId**
  - the unique id/name of the external system wich use this the system doMail
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 50 characters
- **extId**
  - the unique id in external system wich use this the system doMail.
  - the doMail checks if duplicity is enabled/disabled. If ’duplicity’ is disabled so it will not send more emails with the same ’extId’
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 128 characters
- **lastProcessingState**
  - last processing status

  - string
  - one value from values: {RECEIVED, COMPLETE, FAILED, DUPLICATE, PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED, DISPATCH_ERROR, CANCELLED}

- **lastDeliveryState**
  - last delivery status
  - string
  - one value from values: {DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS}

- **scenario**
  - the name of the script to be used for this communication
  - pattern: [-0-9a-zA-Z_@#/́]+
  - maximum length is 255 characters

- **testmode**
  - if the field is set to TRUE, the communication is in test mode
  - default value is FALSE

- **statistics**
  - statistical data that help to classify a given communication
  - **group**
    * pattern: [-0-9a-zA-Z_@#/]+
    * maximum length is 50 characters
  - **operation**
    * pattern: [-0-9a-zA-Z_@#/]+
    * maximum length is 50 characters
  - **category**
    * pattern: [-0-9a-zA-Z_@#/]+
    * maximum length is 50 characters
  - **tags**
    * maximum count of items is 5

- **advancedRequestData** - object type is 'AdvancedRequestType', subfields are:
  - **priority**
    * if we want to send a high priority email, we set the value to TRUE
    * boolean
    * value is true/false
  - **duplicity**

    * if we want to send more emails with the same extId, we have to set the value to TRUE
    * boolean
    * value is true/false
  - **sendTime**
    * If it is set, the email will be sent at that time
    * string (date-time)

- **runs** - object type is 'CommunicationProcDataType', subfields are:
  - **runId**
    * running id of communication
    * integer
    * value have to be greater as 0
  - **procTime**
    * date and time of processing
    * string (date-time)
  - **scenarioName**
    * the name of the script was used for this communication
    * maximum length is 255 characters
  - **processingState**
    * processing status
    * string
    * one value from values: {RECEIVED, COMPLETE, FAILED, DUPLICATE,
      PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED, DISPATCH_ERROR, CANCELLED}
  - **deliveryState**
    * last delivery status
    * string
    * one value from values: {DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS}
  - **sentEmails**
    * more emails can be generated during the processing of the communication, here are the details of each
    * object type is 'SendCommunicationType'
    * subfields are:
    * **addressNumber** (required field)
      · email number in communication processing
      · integer
    * **emailAddress**
      · email address
      · pattern: `[^@]+@[^.]+..+`

    - maximum length is 512 characters
  - **processingState**
    - processing status
    - string
    - one value from values: {RECEIVED, COMPLETE, FAILED, DUPLICATE, PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED, DISPATCH_ERROR, CANCELLED}
  - **deliveryState**
    - last delivery status
    - string
    - one value from values: {DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS}
  - **messageId**
    - message ID
    - maximum length is 80 characters
  - **sendTime**
    - If it is set, the email will be sent at that time
    - string (date-time)

3.1.1.2 /communication/extid/{extId}/detail

- HTTP GET request
- Get details of given communication request by ExternalId
- The input parameter is the ExternalId - extId. ExternalId is the ID provided to the request when calling theMail system.
- Response is also JSON object type of 'CommunicationDetail'. The model of the 'CommunicationDetail' object is shown in the figures 5 and 6.

3.1.1.3 /communication/{id}/state

- HTTP GET request
- Get state of given communication request by Id
- The input parameter is the id. This is the return value when a request is successfully sent to the system DoMail .

In the following figure 7 is the response JSON object of type Ćommunication-StateType:

```
communicationStateType ⌄ {
    lastProcessingState     processingStateType  string
                            Enum:
                             ⌄ [ RECEIVED, COMPLETE, FAILED, DUPLICATE, PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED,
                            DISPATCH_ERROR, CANCELLED ]
    lastDeliveryState       deliveryStateType  string
                            Enum:
                             ⌄ [ DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS ]
    id*                     integer($int64)
    runs                     ⌄ [StateRunType ⌄ {
                                processingState        processingStateType  string
                                                       Enum:
                                                        ⌄ [ RECEIVED, COMPLETE, FAILED, DUPLICATE, PROCESSING, BLACKLISTED,
                                                       DISPATCHING, DISPATCHED, DISPATCH_ERROR, CANCELLED ]
                                deliveryState          deliveryStateType  string
                                                       Enum:
                                                        ⌄ [ DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN,
                                                       DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS ]
                                runId                  integer($int32)
                                sentEmails              ⌄ [StateEmailType ⌄ {
                                                          addressNumber         integer
                                                          addressName           addressName  string
                                                                                maxLength: 200
                                                          emailAddress          emailAddress  string
                                                                                maxLength: 512
                                                                                pattern: [^@]+@[^.]+..+
                                                          processingState       processingStateType  string
                                                                                Enum:
                                                                                 > Array [ 10 ]
                                                          deliveryState         deliveryStateType  string
                                                                                Enum:
                                                                                 > Array [ 5 ]

                                                         }]

                             }]

}
```

Figure 7: Model of object CommunicationStateType

Object of type ’CommunicationStateType’ has fields:

- **lastProcessingState**
  - last processing status
  - string
  - one value from values: {RECEIVED, COMPLETE, FAILED, DUPLICATE, PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED, DISPATCH_ERROR, CANCELLED}
- **lastDeliveryState**
  - last delivery status
  - string
  - one value from values: {DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS}
- **id** (required field)
  - main id of communication

- **runs** - object type is 'CommunicationProcDataType', subfields are:
  - **runId**
    - running id of communication
    - integer
    - value have to be greater as 0
  - **procTime**
    - date and time of processing
    - string (date-time)
  - **scenarioName**
    - the name of the script was used for this communication
    - maximum length is 255 characters
  - **processingState**
    - processing status
    - string
    - one value from values: {RECEIVED, COMPLETE, FAILED, DUPLICATE,
      PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED, DISPATCH_ERROR, CANCELLED}
  - **deliveryState**
    - last delivery status
    - string
    - one value from values: {DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS}
  - **sentEmails**
    - more emails can be generated during the processing of the communication, here are the details of each
    - object type is 'SendCommunicationType'
    - subfields are:
    - **addressNumber** (required field)
      - email number in communication processing
      - integer
    - **emailAddress**
      - email address
      - pattern: `[^@]+@[^.]+..+`
      - maximum length is 512 characters
    - **processingState**
      - processing status
      - string
      - one value from values: {RECEIVED, COMPLETE, FAILED, DUPLICATE, PROCESSING, BLACKLISTED, DISPATCHING, DISPATCHED, DISPATCH_ERROR, CANCELLED}
    - **deliveryState**

        - last delivery status
        - string
        - one value from values: {DELIVERY_FAILED, DELIVERY_NONE, DELIVERY_UNKNOWN, DELIVERY_CONFIRMED, DELIVERY_AMBIGUOUS}
    - **messageId**
        - message ID
        - maximum length is 80 characters
    - **sendTime**
        - If it is set, the email will be sent at that time
        - string (date-time)

3.1.1.4 /communication/extid/{extId}/state

- HTTP GET request
- Get state of given communication request by ExternalId
- The input parameter is the ExternalId - extId. ExternalId is the ID provided to the request when calling theMail system.
- Response is also JSON object type of 'CommunicationStateType'. The model of the 'CommunicationStateType' object is shown in the figure 7.

3.1.2 Email

The following endpoints are available in email:

- POST /email/send
- POST /email/sendAdvanced
- POST /email/sendWithTemplate
- POST /email/sendWithTemplateAdvanced

3.1.2.1 /email/send

- HTTP POST request,
- Endpoint is used to simply send an email via REST API,
- Body of POST request is JSON object type of 'emailSendRequest',
- Response is object type of 'CommunicationId'

In the following figure 8 is the definition of the JSON object 'emailSendRequest':

```
emailSendRequest ⌄ {
    to*                     ⌄ [
                            maxItems: 100
                            emailAddressType ⌄ {
                                addressName         addressName  string
                                                    maxLength: 200
                                emailAddress*       emailAddress  string
                                                    maxLength: 512
                                                    pattern: [^@]+@[^.]+..+

                             }]
    subject                 string255  string
                            maxLength: 255
    htmlBody                string
    plainTextBody           string
    scenario                std255Regex  string
                            pattern: [-0-9a-zA-Z_@#/]+
                            maxLength: 255
    testmode                boolean
                            default: false
    attachments             ⌄ [
                            maxItems: 20
                            fileHandlerType ⌄ {
                                filename*           string255  string
                                                    maxLength: 255
                                contentID           string
                                mimeType            string
                                encoding            string
                                dataHandler*        string($byte)
                                                    pattern: ^(?:[A-Za-z0-9+/]{4})*(?:[A-Za-z0-9+/]{2}==|[A-Za-z0-9+/]{3}=)?$
                                                    maxLength: 30720000
                                                    xml: OrderedMap { "name": "dataHandler", "attribute": false, "wrapped": false }
                                                    xml:
                                                       name: dataHandler
                                                       attribute: false
                                                       wrapped: false

                             }]
    statistics              statisticsType ⌄ {
                                group               std50Regex  string
                                                    pattern: [-0-9a-zA-Z_@#/]+
                                                    maxLength: 50
                                operation           std50Regex  string
                                                    pattern: [-0-9a-zA-Z_@#/]+
                                                    maxLength: 50
                                category            std50RegexOpt  string
                                                    maxLength: 50
                                                    pattern: [-0-9a-zA-Z_@#/]*
                                tags                ⌄ [
                                                    maxItems: 5
                                                    std50Regex  string
                                                    pattern: [-0-9a-zA-Z_@#/]+
                                                    maxLength: 50]

                            }
}
```

Figure 8: Object type of ’emailSendRequest’

- **to** (required field)
  - list of email address of recipients
  - minimum count of recipients is 1 and maximum 100 recipients
- **subject**
  - subject of email
  - maximum length is 255 characters
- **htmlBody**

  - html text body of email
  - string
- **plainTextBody**
  - plain text body of email
  - string
- **scenario**
  - the name of the script to be used for this communication
  - pattern: [-0-9a-zA-Z_@#/́]+
  - maximum length is 255 characters
- **testmode**
  - if the field is set to TRUE, the communication is in test mode
  - default value is FALSE
- **attachments**
  - array of attachments
  - maximum count of items is 20
  - One item of type **fileHandlerType**:
    - **filename** (required field)
      - file name
      - maximum length is 255 characters
    - **contentId**
      - Content ID
      - string
    - **mimeType**
      - MimeType (Example: application/text)
      - string
    - **encoding**
      - Encoding (Example: utf-8)
      - string
    - **dataHandler** (required field)
      - Base64 string
- **statistics**
  - statistical data that help to classify a given communication
  - **group**
    - pattern: [-0-9a-zA-Z_@#/]+
    - maximum length is 50 characters
  - **operation**
    - pattern: [-0-9a-zA-Z_@#/]+
    - maximum length is 50 characters

- **category**
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 50 characters
- **tags**
  - maximum count of items is 5

In the following figure 9 is the definition of the JSON object ’CommunicationId’:

```
communicationId ⌄ {
    id*                    integer($int64)

}
```

Figure 9: Object type of ’CommunicationId’

Response JSON object ’CommunicationId’:

- **id**
  - Id of send communication in the system doMail,
  - It has to be bigger than 0

3.1.2.2 /email/sendAdvanced

- HTTP POST request,
- Send email (advanced),
- Body of POST request is JSON object type of ’EmailSendAdvancedRequest’,
- Response is object type of ’CommunicationId’

Object type of ’EmailSendAdvancedRequest’ (10) is inherited from object type of ’EmailSendRequest’. ’EmailSendAdvancedRequest’ contains all fields from object type ’EmailSendAdvancedRequest’ (green part of from picture 10).

```
emailSendRequest ⌄ {
    to*                  > [...]
    subject              string255 > [...]
    htmlBody             > [...]
    plainTextBody        > [...]
    scenario             std255Regex > [...]
    testmode             > [...]
    attachments          > [...]
    statistics           statisticsType ⌄ {
                             group        std50Regex > [...]
                             operation    std50Regex > [...]
                             category     std50RegexOpt > [...]
                             tags         > [...]
                         }
    sysId*               std50Regex  string
                         pattern: [-0-9a-zA-Z_@#/]+
                         maxLength: 50
    extId*               std128Regex  string
                         pattern: [-0-9a-zA-Z_@#/]+
                         maxLength: 128
    advancedRequestData  advancedRequestType ⌄ {
                             priority     boolean
                             duplicity    boolean
                             sendTime     string($date-time)
                         }
    additionalContent    ⌄ [additionalContentType ⌄ {
                             key          std50Regex  string
                                          pattern: [-0-9a-zA-Z_@#/]+
                                          maxLength: 50
                             value        ⌄ {
                                          }
                         }]
}
```

Figure 10: Object type of ’EmailSendAdvancedRequest’

New fields (red part of from picture 10) are:

- **sysId**
  - the unique id/name of the external system wich use this the system doMail
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 50 characters
- **extId**
  - the unique id in external system wich use this the system doMail.
  - the doMail checks if duplicity is enabled/disabled. If 'duplicity' is disabled so it will not send more emails with the same 'extId'
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 128 characters
- **advancedRequestData** - object type is 'AdvancedRequestType', subfields are:
  - **priority**
    * if we want to send a high priority email, we set the value to TRUE
    * boolean
    * value is true/false
  - **duplicity**
    * if we want to send more emails with the same extId, we have to set the value to TRUE
    * boolean
    * value is true/false
  - **sendTime**
    * If it is set, the email will be sent at that time
    * string (date-time)
    * value have to be time greater than now
- **additionalContent** - array of objects type 'additionalContentType'. Subfields of object type 'additionalContentType' are:
  - **key**
    * Key is unique name of item
    * pattern: [-0-9a-zA-Z_@#/]+
    * maximum length is 50 characters
  - **value**
    * Value of item
    * object

3.1.2.3 /email/sendWithTemplate

- HTTP POST request
- Send of email with template
- Body of POST request is JSON object type of 'EmailSendWithTemplateRequest',
- Response is object type of 'CommunicationTemplateId'

In the following figure 11 is the definition of the JSON object 'EmailSendWithTemplateRequest':

```
emailSendWithTemplateRequest ⌄ {
    scenario*           std255Regex  string
                        pattern: [-0-9a-zA-Z_@#/]+
                        maxLength: 255
    statistics          statisticsType ⌄ {
                           group            std50Regex  string
                                            pattern: [-0-9a-zA-Z_@#/]+
                                            maxLength: 50
                           operation        std50Regex  string
                                            pattern: [-0-9a-zA-Z_@#/]+
                                            maxLength: 50
                           category         std50RegexOpt  string
                                            maxLength: 50
                                            pattern: [-0-9a-zA-Z_@#/]*
                           tags             ⌄ [
                                            maxItems: 5
                                            std50Regex  string
                                            pattern: [-0-9a-zA-Z_@#/]+
                                            maxLength: 50]
                        }
    testmode            boolean
                        default: false
    params              ⌄ [parametersTemplate ⌄ {
                           key              std50Regex  string
                                            pattern: [-0-9a-zA-Z_@#/]+
                                            maxLength: 50
                           value            string1024  string
                                            maxLength: 1024
                        }]
    attachments         ⌄ [
                        maxItems: 20
                        fileHandlerType ⌄ {
                           filename*        string255  string
                                            maxLength: 255
                           contentID        string
                           mimeType         string
                           encoding         string
                           dataHandler*     string($byte)
                                            pattern: ^(?:[A-Za-z0-9+/]{4})*(?:[A-Za-z0-9+/]{2}==|[A-Za-
                                            z0-9+/]{3}=)?$
                                            maxLength: 30720000
                                            xml: OrderedMap { "name": "dataHandler", "attribute": false,
                                            "wrapped": false }
                                            xml:
                                               name: dataHandler
                                               attribute: false
                                               wrapped: false
                        }]
}
```

Figure 11: Object type of 'EmailSendWithTemplateRequest'

Properties of object type 'EmailSendWithTemplateRequest':

- **scenario** (required field)
  - the name of the script to be used for this communication
  - pattern: [-0-9a-zA-Z_@#/́]+

  - maximum length is 255 characters

- **statistics** - JSON object type of 'StatisticsType'
  - statistical data that help to classify a given communication
  - **group**
    - pattern: [-0-9a-zA-Z_@#/]+
    - maximum length is 50 characters
  - **operation**
    - pattern: [-0-9a-zA-Z_@#/]+
    - maximum length is 50 characters
  - **category**
    - pattern: [-0-9a-zA-Z_@#/]+
    - maximum length is 50 characters
  - **tags**
    - maximum count of items is 5
    - pattern: [-0-9a-zA-Z_@#/]+

- **testmode**
  - if the field is set to TRUE, the communication will be to send in test mode
  - default value is FALSE

- **params**
  - list of parameters to be used in the template
  - array of objects type of 'ParametersTemplate'
  - **key**
    - string
    - pattern: [-0-9a-zA-Z_@#/]+
  - **value**
    - string
    - maximum length is 1024 characters

- **attachments**
  - array of attachments
  - maximum count of items is 20
  - One item of type **fileHandlerType**:
    - **filename** (required field)
      - file name
      - maximum length is 255 characters
    - **contentId**
      - Content ID
      - string
    - **mimeType**

        - MimeType (Example: application/text)
        - string
    - **encoding**
        - Encoding (Example: utf-8)
        - string
    - **dataHandler** (required field)
        - Base64 string

In the following figure 12 is the definition of the JSON object 'CommunicationTemplateId':

```
communicationTemplateId ⌄ {
    mainCommunicationId  integer($int64)
    subCommunicationId
                         ⌄ [integer($int64)]
}
```

Figure 12: Object type of 'CommunicationTemplateId'

Response JSON object 'CommunicationTemplateId':

- **mainCommunicationId**
    - id of the main meta communication,
    - It has to be bigger than 0
- **subCommunicationId** - array of type integer 64bit (long)
    - List of subcommunication IDs that were created based on the defined template

Now follows the HTTP POST example (14) to call endpoint:
http:\\{yourUrl}/domail-input-rest/email/sendWithTemplate

```
{
    "scenario": "testTemplate"
}
```

Listing 14: Example for POST request for sending email with template only with required fields

After successful sending, the response will arrive as it is in 15.

```
{
    "mainCommunicationId": 1438,
    "subCommunicationId": []
}
```

Listing 15: Example for POST request for sending email with template only with required fields

3.1.2.4 /email/sendWithTemplateAdvanced

- HTTP POST request
- Send of email with template (Advanced)
- Body of POST request is JSON object type of 'EmailSendWithTemplateAdvancedRequest',
- Response is object type of 'CommunicationTemplateId'

In the following figure 13 is the definition of the JSON object 'EmailSendWithTemplateAdvancedRequest':

```
emailSendWithTemplateAdvancedRequest ⌄ {
    scenario*            std255Regex  string
                         pattern: [-0-9a-zA-Z_@#/]+
                         maxLength: 255
    sysId*               std50Regex  string
                         pattern: [-0-9a-zA-Z_@#/]+
                         maxLength: 50
    extId*               std128Regex  string
                         pattern: [-0-9a-zA-Z_@#/]+
                         maxLength: 128
    statistics           statisticsType ⌄ {
                             group                std50Regex  string
                                                  pattern: [-0-9a-zA-Z_@#/]+
                                                  maxLength: 50
                             operation            std50Regex  string
                                                  pattern: [-0-9a-zA-Z_@#/]+
                                                  maxLength: 50
                             category             std50RegexOpt  string
                                                  maxLength: 50
                                                  pattern: [-0-9a-zA-Z_@#/]*
                             tags                  ⌄ [
                                                  maxItems: 5
                                                  std50Regex  string
                                                  pattern: [-0-9a-zA-Z_@#/]+
                                                  maxLength: 50]
                         }
    testmode             boolean
                         default: false
    advancedRequestData  advancedRequestType ⌄ {
                             priority             boolean
                             duplicity            boolean
                             sendTime             string($date-time)
                         }
    attachments           ⌄ [
                         maxItems: 50000
                         fileHandlerType ⌄ {
                             filename*            string255 › [...]
                             contentID            string
                             mimeType             string
                             encoding             string
                             dataHandler*         string($byte)
                                                  pattern: ^(?:[A-Za-z0-9+/]{4})*(?:[A-Za-z0-9+/]{2}==|[A-Za-z0-9+/]
                                                  {3}=)?$
                                                  maxLength: 30720000
                                                  xml: OrderedMap { "name": "dataHandler", "attribute": false,
                                                  "wrapped": false }
                                                  xml:
                                                     name: dataHandler
                                                     attribute: false
                                                     wrapped: false
                          }]
    templateDataJSON     templateDataJSON ⌄ [ ⌄ {
                          }]
    templateDataCSV      templateDataCSV ⌄ [fileHandlerType › {...}]
}
```

Figure 13: Object type of 'EmailSendWithTemplateAdvancedRequest'

Properties of object type 'EmailSendWithTemplateAdvancedRequest':

- **scenario** (required field)
  - the name of the script to be used for this communication

  - pattern: [-0-9a-zA-Z_@#/́]+
  - maximum length is 255 characters

- **sysId** (required field)
  - the unique id/name of the external system wich use this the system doMail
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 50 characters

- **extId** (required field)
  - the unique id in external system wich use this the system doMail.
  - the doMail checks if duplicity is enabled/disabled. If 'duplicity' is disabled so it will not send more emails with the same 'extId'
  - pattern: [-0-9a-zA-Z_@#/]+
  - maximum length is 128 characters

- **statistics** - JSON object type of 'StatisticsType'
  - statistical data that help to classify a given communication
  - **group**
    * pattern: [-0-9a-zA-Z_@#/]+
    * maximum length is 50 characters
  - **operation**
    * pattern: [-0-9a-zA-Z_@#/]+
    * maximum length is 50 characters
  - **category**
    * pattern: [-0-9a-zA-Z_@#/]+
    * maximum length is 50 characters
  - **tags**
    * maximum count of items is 5
    * pattern: [-0-9a-zA-Z_@#/]+

- **testmode**
  - if the field is set to TRUE, the communication will be to send in test mode
  - default value is FALSE

- **advancedRequestData** - object type is 'AdvancedRequestType', subfields are:
  - **priority**
    * if we want to send a high priority email, we set the value to TRUE
    * boolean
    * value is true/false
  - **duplicity**

        * if we want to send more emails with the same extId, we have to set the value to TRUE
        * boolean
        * value is true/false
    - **sendTime**
        * If it is set, the email will be sent at that time
        * string (date-time)
        * value have to be time greater than now

- **attachments**
    - array of attachments
    - maximum count of items is 20
    - One item of type **fileHandlerType**:
        * **filename** (required field)
            · file name
            · maximum length is 255 characters
        * **contentId**
            · Content ID
            · string
        * **mimeType**
            · MimeType (Example: application/text)
            · string
        * **encoding**
            · Encoding (Example: utf-8)
            · string
        * **dataHandler** (required field)
            · Base64 string

- **templateDataJSON**
    - array of any objects

- **templateDataCSV**
    - object of type **fileHandlerType** - same as was in attachments:
        * **filename** (required field)
            · file name
            · maximum length is 255 characters
        * **contentId**
            · Content ID
            · string
        * **mimeType**
            · MimeType (Example: application/text)
            · string
        * **encoding**

        - Encoding (Example: utf-8)
        - string
    - **dataHandler** (required field)
        - Base64 string

3.2 Create OpenAPI client in C#

In the following example 16 there is an object of type CustomOpenApiClient, which is inherited from the generated object of type 'Client'. The generated object 'Client' was created based on the definition written in the YAML file that describes the REST API. The inherited object 'CustomOpenApiClient' is needed to change the settings in 'JsonSerializerSettings.DateFormatString' to "yyyyMMddTHH:mm:ss.fffffffZ" and JsonSerializerSettings.DateTimeZoneHandling to Newtonsoft.Json.DateTimeZoneHandling.Local, because the system doMail works with the local time of the server.

```
public class CustomOpenApiClient : Client
{
    public CustomOpenApiClient(string baseUrl) : base(baseUrl, new HttpClient())
    {
        this.UpdateSerializerSettings();
    }


    private void UpdateSerializerSettings()
    {
        //2023-04-14T14:39:55.9794317
        base.JsonSerializerSettings.DateFormatString = "yyyy-MM-ddTHH:mm:ss.fffffffZ";

        //base.JsonSerializerSettings.DateFormatHandling = Newtonsoft.Json.
    DateFormatHandling.MicrosoftDateFormat;
        base.JsonSerializerSettings.DateTimeZoneHandling = Newtonsoft.Json.
    DateTimeZoneHandling.Local;

    }
}
```

Listing 16: Object type of 'CustomOpenApiClient' inherited from 'Client'

4 Change state notification queues

Domail supports notifications via Apache ActiveMQ, which allows for efficient and reliable message queuing. The web interface for Apache ActiveMQ can be accessed at DOMAIL_URL:8162. This interface provides comprehensive monitoring and management capabilities for all message queues used within the Domail system.
The access credentials for Apache ActiveMQ are configured in the /opt/nuncio/domail/domail_amq/conf/jetty-realm.properties file. By default, the login credentials are:

Administrator

- Username: ‘admin‘
- Password: ‘admin‘

User

- Username: ‘user‘
- Password: ‘user‘

It is highly recommended to change these default passwords to ensure the security of your system.

4.1 Key Features and Monitoring Capabilities

The Apache ActiveMQ web interface offers several features that allow users to monitor and manage the message queues effectively. Some of the key features include:

- Viewing Queues: Users can view all the available queues in the system. This includes queues for both outgoing and incoming emails.
- Monitoring Messages: Users can see the messages waiting in each queue. This helps in understanding the current load and identifying any potential bottlenecks.
- Processing Workers: The interface provides information on the number of processors working on each queue. This helps in assessing the processing capacity and performance.

4.2 Important Queues

Several queues are particularly important for monitoring within the Domail system. These queues include:

- COM_MAIL_priority: This queue handles priority outgoing email messages. Monitoring this queue ensures that high-priority emails are being processed and sent promptly.
- COM_MAIL_normal: This queue manages normal priority outgoing email messages. Keeping an eye on this queue helps ensure that regular emails are processed efficiently.

- COM_IN_priority: This queue is for incoming email messages that are marked as priority. Monitoring this queue ensures that important incoming emails are processed quickly.
- COM_IN_normal: This queue handles normal priority incoming email messages. This queue should be monitored to ensure that all incoming emails are being processed appropriately.

Queue name	Pending messages	Number of consumers
COM_NOTIF	8417	
COM_MAIL_priority		1
COM_COARARCH	1139	
COM_MAIL_normal		3
COM_IN_priority		1
COM_IN_normal		2
Domic.COM_notif		

Figure 14: Queues list

4.3 Using the Domail Web Interface

Through the Domail web interface, users can:

- Access All Queues: View and manage all message queues within the system.
- Check Pending Messages: See the number of messages pending in each queue, helping to understand the system load and efficiency.
- Monitor Processing Workers: View the number of workers processing messages in each queue to ensure there are enough resources allocated for efficient processing.

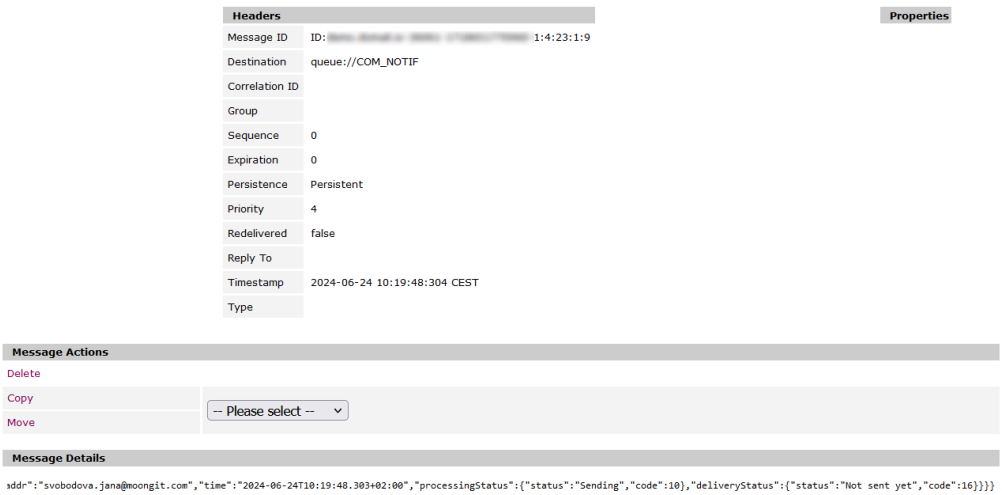

Figure 15: ActiveMQ detail

Domail uses ActiveMQ to store every change in the status of communications. These status changes can be tracked not only through the ActiveMQ interface but also in the communication details section under "Tasks". This dual tracking ensures comprehensive monitoring and easier troubleshooting of communication processes. Domail reports the entire lifecycle of processing and delivering communications to ActiveMQ queues. If needed, multiple queues can be utilized. The appropriate queue is selected based on the cc_inst field (also known as instance id) of the request. Consider an example with two separate organizational units - East and West - using a single Domail instance. These two units can differentiate their emails and requests by using different cc_inst fields: 'east' and 'west'. In the application.properties file, you can specify:

```
app.mq.notifications.east = QueueNameEast
app.mq.notifications.west = QueueNameWest
```

Listing 17: Example of two instances

4.4 Notification Messages

Notifications are JSON messages specified by a predefined JSON schema. These messages contain three main parts:

- Header: Contains communication ID, run ID, and statistical and categorization fields like group, operation, category, tags, etc.
- Request Processing: Information about the processing run, including state and delivery state.
- Email: Information about the email, including address, state, and delivery state.

Predefined JSON schema of notification (listing 18):

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Domail notification",
    "description": "Schema for immediate state notification of communications/emails",
    "type": "object",
    "required": ["notification"],
    "properties": {
      "notification": {
        "type": "object",
        "required": ["notificationTime", "header"],
        "properties": {
          "notificationTime": { "type": "string", "format": "date-time" },
          "header": {
            "type": "object",
            "required": ["requestKey"],
            "properties": {
              "requestKey": {
                "type": "object",
                "properties": {
                  "instanceID": { "type": "string" },
                  "sysId": { "type": "string" },
                  "extId": { "type": "string" },
                  "scenario": { "type": "string" },
                  "operation": { "type": "string" },
                  "category": { "type": "string" },
                  "domailID": { "type": "integer" },
                  "domailRunID": { "type": "integer" }
                },
                "required": [
                  "instanceID",
```

```
                      "sysId",
                      "extId",
                      "scenario",
                      "operation",
                      "category",
                      "domailID",
                      "domailRunID"
                    ]
                  },
                  "tags": { "type": "string" }
                }
              },
              "requestProcessing": {
                "type": "object",
                "properties": {
                  "time": { "type": "string", "format": "date-time" },
                  "processingStatus": {
                    "type": "object",
                    "properties": {
                      "status": {
                        "type": "string",
                        "enum": [
                          "Request received",
                          "Request processed",
                          "Error",
                          "Duplicate",
                          "Request processing",
                          "Cancelled"
                        ]
                      },
                      "code": { "type": "integer", "enum": [2, 3, 4, 6, 8, 13] },
                      "message": { "type": "string" }
                    },
                    "required": ["status", "code"]
                  },
                  "deliveryStatus": {
                    "type": "object",
                    "properties": {
                      "status": {
                        "type": "string",
                        "enum": [
                          "Not sent yet",
                          "Delivery unknown",
                          "Delivery confirmed",
                          "Delivery failed",
                          "Delivery ambiguous"
                        ]
                      },
                      "code": { "type": "integer", "enum": [16, 17, 18, 15, 20] },
                      "message": { "type": "string" }
                    },
                    "required": ["status", "code"]
                  },
                  "coarArchivationStatus": {
                    "type": "object",
                    "properties": {
                      "status": {
                        "type": "string",
                        "enum": [
                          "Not archived in COAR",
                          "Archived in COAR",
                          "Archivation to COAR",
                          "Archivation to COAR failed"
                        ]
                      },
                      "code": { "type": "integer", "enum": [0, 7, 21, 22] }
                    },
                    "required": ["status", "code"]
                  }
                },
                "required": ["time", "processingStatus", "deliveryStatus"]
              },
              "email": {
                "type": "object",
```

```
                "properties": {
                  "number": { "type": "integer" },
                  "addr": { "type": "string" },
                  "time": { "type": "string", "format": "date-time" },
                  "processingStatus": {
                    "type": "object",
                    "properties": {
                      "status": {
                        "type": "string",
                        "enum": [
                          "Request received",
                          "Request processed",
                          "Error",
                          "Request processing",
                          "Cancelled",
                          "Email blocked",
                          "Sending failure",
                          "Sending",
                          "Sent",
                          "Duplicate"
                        ]
                      },
                      "code": {
                        "type": "integer",
                        "enum": [2, 3, 4, 8, 13, 9, 12, 10, 11, 6]
                      },
                      "message": { "type": "string" }
                    },
                    "required": ["status", "code"]
                  },
                  "deliveryStatus": {
                    "type": "object",
                    "properties": {
                      "status": {
                        "type": "string",
                        "enum": [
                          "Not sent yet",
                          "Delivery unknown",
                          "Delivery confirmed",
                          "Delivery failed"
                        ]
                      },
                      "code": { "type": "integer", "enum": [16, 17, 18, 15] },
                      "smtpMessage": { "type": "string" },
                      "smtpCode": { "type": "integer" }
                    },
                    "required": ["status", "code"]
                  }
                },
                "required": [
                  "number",
                  "addr",
                  "time",
                  "processingStatus",
                  "deliveryStatus"
                ]
              }
            }
          }
        }
      }
```

Listing 18: JSON schema of notification

In the early stages of processing, information is aggregated in the run, and individual emails are not reported. In later stages, each email is processed individually, and the run is not reported. By using this approach, Domail ensures that all relevant information about communication processing and delivery is accurately tracked and reported to the appropriate ActiveMQ queues, allowing for comprehensive monitoring and analysis.

5 Scripts

In these examples we will show how different programming languages can be used to send an email (create a communication) or to get details about the requested communication.

5.1 Example 1 - Send email through REST API - EmailSend-Simple

5.1.1 JSON

Now follows a simple HTTP POST example to call endpoint:
http:\\{yourUrl}/domail-input-rest/email/send

We can test this e.g. via Postman.

```
{
    "to": [
        {
          "addressName": "NameOfEmail",
          "emailAddress": "email@email.com"
        }
    ]
}
```

Listing 19: Example for POST request for sending email (simple) only with required fields

Alternatively, it can be called using the command CURL in next listing 20, now URL for doMail system is http://domail-test/ :

```
    {
    "to": [
        {
            "addressName": "NameOfEmail",
            "emailAddress": "email@email.com"
        }
    ]
}
```

Listing 20: Example for POST request for sending email (simple) only with required fields with command CURL

5.1.2 C#

Another language is C#, in which we used an object of type 'CustomOpenApiClient' created by us, which is inherited from the object of type 'Client'.

At the beginning we need to initialize an instance of the 'CustomOpenApiClient' object, through which we connect to the doMail system.

```
//Create CustomOpenApiClient
string url = $"{Configuration.DOMAIL_URL}/domail-input-rest/";
var client = new CustomOpenApiClient(url);
```

Next, we will prepare an object of type 'EmailSendRequest', which will contain all the information about the request sent to the server. We fill the same fields as we filled in the example in JSON.

```
//Create EmailSendRequest
EmailSendRequest emailRequest = new EmailSendRequest();

```

```
//set 1 recipients
emailRequest.To = new List<EmailAddressType>() {
        new EmailAddressType() { AddressName = "NameOfEmail", EmailAddress = "email@email.com"}
};
```

Now the prepared request is sent via the ASYNC call provided by the system doMail. Since we want to wait for a response, we wait for processing to print the return.

```
//send reqest throug REST API EmailSendSimpleAsync
Task<CommunicationId> taskEmailSendSimpleAsync = client.EmailSendSimpleAsync(emailRequest);

//wait for resposne
var result = taskEmailSendSimpleAsync.GetAwaiter().GetResult();

if (result.Id > 0)
    Console.WriteLine($"call POST OK - result.Id={result.Id}");
```

```
try
{
    ...
}
catch (ApiException<RestErrorResponse> e1)
{
    Console.WriteLine($"ApiException: {e1}\nStatusCode: {e1.StatusCode}\nResult.ResponseCode: {e1.Result?.ResponseCode}\nResult.Message: {e1.Result?.Message}");
}
catch (ApiException e2)
{
    Console.WriteLine($"ApiException: {e2}, StatusCode: {e2.StatusCode}");
}
catch (Exception ex)
{
    Console.WriteLine($"Exception: {ex}");
}
```

Complete source file is on the next listening 21

```
try
{
    //Create CustomOpenApiClient
    string url = $"{Configuration.DOMAIL_URL}/domail-input-rest/";
    var client = new CustomOpenApiClient(url);

    //Create EmailSendRequest
    EmailSendRequest emailRequest = new EmailSendRequest();

    //set 1 recipients
    emailRequest.To = new List<EmailAddressType>() {
            new EmailAddressType() { AddressName = "NameOfEmail", EmailAddress = "email@email.com"}
    };


    //send reqest throug REST API EmailSendSimpleAsync
    Task<CommunicationId> taskEmailSendSimpleAsync = client.EmailSendSimpleAsync(emailRequest);

    //wait for resposne
    var result = taskEmailSendSimpleAsync.GetAwaiter().GetResult();
```

```
    if (result.Id > 0)
        Console.WriteLine($"call POST OK - result.Id={result.Id}");

}
catch (ApiException<RestErrorResponse> e1)
{
    Console.WriteLine($"ApiException: {e1}\nStatusCode: {e1.StatusCode}\nResult.
     ResponseCode: {e1.Result?.ResponseCode}\nResult.Message: {e1.Result?.Message}");
}
catch (ApiException e2)
{
    Console.WriteLine($"ApiException: {e2}, StatusCode: {e2.StatusCode}");
}
catch (Exception ex)
{
    Console.WriteLine($"Exception: {ex}");
}
```

Listing 21: Example for POST request for sending email (simple) only with required fields in C#

5.1.3 Java

Another language is Java. This improved Java program demonstrates how to send an email using the HttpURLConnection class to interact with a REST API. The program constructs a JSON request body with email details and sends a POST request to the specified URL. Basic Authentication is used for API access.

5.1.3.1 Import Statements

These import statements include the necessary classes for handling network connections, input/output streams, and JSON manipulation.

```
    import java.io.InputStream;
    import java.io.OutputStream;
    import java.net.HttpURLConnection;
    import java.net.URL;
    import java.nio.charset.StandardCharsets;
    import java.util.Base64;
    import java.util.Scanner;
    import org.json.JSONArray;
    import org.json.JSONObject;
```

Listing 22: Import Statements

5.1.3.2 Main Class and Method

Main Method: The main method calls sendEmail and handles any exceptions.

```
    public class Main {

    private static final String SERVICE_URL = "{Configuration.DOMAIL_URL}/domail-input-
     rest/email/send";
    private static final String USER_CREDENTIALS = "name:XXXXXXXXXX";
    private static final String SCENARIO = "Default_REST";
    private static final String SUBJECT = "Test sending email via Java REST.";
    private static final String PLAIN_TEXT_BODY = "Hello, this is test of sending email
     via Java.";
```

```java

    public static void main(String[] args) {
        try {
            sendEmail();
        } catch (Exception e) {
            System.err.println("Error sending email: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

Listing 23: Main Class and Method

5.1.3.3 'sendEmail' Method

- URL Initialization: The URL of the email service is defined using the SERVICE_URL constant.
- JSON Object Creation: A JSONObject is created to represent the email, and a JSONArray is used to hold the recipient details.
- HttpURLConnection Setup: The HttpURLConnection object is created and configured for a POST request.
- Basic Authentication: Basic Authentication is set up using the encoded user credentials.
- Request Headers: Request headers are set to indicate that the content type is JSON.
- Output Stream: The JSON request body is written to the output stream using UTF-8 encoding.
- Response Handling: The response code and message are printed. The input stream is read to get the response body. If an error stream is available, it is used instead.
- Resource Management: Try-with-resources is used to ensure the output stream and scanner are properly closed.
- Exception Handling: Specific exceptions are caught and printed to the console.

```java
    private static void sendEmail() throws Exception {
        // Define the URL of the service
        URL url = new URL(SERVICE_URL);
        JSONArray arr = new JSONArray();
        arr.put(new JSONObject().put("addressName", "Recipient").put("emailAddress", "
    recipient@example.com"));

        // Create a JSON object with email details
        JSONObject json = new JSONObject();
        json.put("scenario", SCENARIO);
        json.put("to", arr);
        json.put("testmode", false);
        json.put("subject", SUBJECT);
        json.put("plainTextBody", PLAIN_TEXT_BODY);

        // Open a connection to the URL
```

```java
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();

        // Set the request method to POST
        connection.setRequestMethod("POST");

        // Setting up Basic Authentication
        String basicAuth = "Basic " + Base64.getEncoder().encodeToString(
    USER_CREDENTIALS.getBytes(StandardCharsets.UTF_8));
        connection.setRequestProperty("Authorization", basicAuth);

        // Set the request headers
        connection.setRequestProperty("Content-Type", "application/json; utf-8");
        connection.setRequestProperty("Accept", "application/json");

        // Enable input and output streams
        connection.setDoOutput(true);

        // Write the JSON request body to the output stream
        try (OutputStream os = connection.getOutputStream()) {
            byte[] input = json.toString().getBytes(StandardCharsets.UTF_8);
            os.write(input, 0, input.length);
        }

        // Get the response code and message
        int responseCode = connection.getResponseCode();
        System.out.println("Response Code: " + responseCode + " " + connection.
    getResponseMessage());

        // Read the response from the input stream
        InputStream stream = connection.getErrorStream();
        if (stream == null) {
            stream = connection.getInputStream();
        }

        try (Scanner scanner = new Scanner(stream)) {
            scanner.useDelimiter("\\Z");
            System.out.println(scanner.next());
        }

        // Disconnect the connection
        connection.disconnect();
    }
}
```

Listing 24: 'sendEmail' Method

5.1.4 Python

To send an email using Python and the requests library to interact with a domail server, follow these steps:

5.1.4.1 Import Statements

These import statements include the necessary classes for sending HTTP requests and handling Basic Authentication.

```python
    import requests
    from requests.auth import HTTPBasicAuth
```

Listing 25: Import Statements

5.1.4.2 Define URL and Credentials

- url: The URL endpoint of the email sending service.
- user and password: The credentials for Basic Authentication.

```
# Define the URL of the email sending service
url = 'https://{{yourUrl}}/domail-input-rest/email/send'

# User credentials for Basic Authentication
user = 'username'
password = 'XXXXXXXXXX'
```

Listing 26: Define URL and Credentials

5.1.4.3 Create Data Payload

- plainTextBody: The body of the email in plain text.
- scenario: The scenario to be used.
- subject: The subject of the email.
- testmode: A boolean indicating whether to use test mode.
- to: A list of recipient details, each containing addressName and emailAddress.

```
# Data payload to be sent in the POST request
data = {
    "plainTextBody": "Hello, this is test of sending email via Python.",
    "scenario": "Default_REST",
    "subject": "Test sending email via Python REST.",
    "testmode": False,
    "to": [
        {
            "addressName": "Recipient",
            "emailAddress": "recipient@example.com"
        }
    ]
}
```

Listing 27: Create Data Payload

5.1.4.4 Send POST Request

- The requests.post method sends a POST request to the specified URL with the given JSON data and Basic Authentication.
- response.raise_for_status() checks if the request was successful and raises an exception if an error occurred.
- The response status code and content are printed.
- In case of an exception (e.g., network error, authentication error), it is caught and printed.

```python
    try:
        # Send a POST request with the JSON data and Basic Authentication
        response = requests.post(url, json=data, auth=HTTPBasicAuth(user, password))

        # Check if the request was successful
        response.raise_for_status()

        # Print the response status code and content
        print(f"Response Code: {response.status_code}")
        print("Response Content:", response.json())

    except requests.exceptions.RequestException as e:
        # Print the error if the request fails
        print(f"An error occurred: {e}")
```

Listing 28: Send POST Request

5.2 Example 2 - Send email through REST API - EmailSend-Simple

The next example is an extension of the previous Example 1 , now we want to send an email with all fields

5.2.1 JSON

In the next example 29, there is POST request with all fields:

```json
  {
    "to": [
     {
          "addressName": "NameOfEmail1",
          "emailAddress": "email1@email.com"
        },
     {
          "addressName": "NameOfEmail2",
          "emailAddress": "email2@email.com"
        }
  ],
  "attachments": [
    {
      "filename": "testFile1",
      "dataHandler": "VG90byBqZSB0ZXN0b3ZhY2lhIHByaWxvaGE=",
      "contentID": "54321",
      "mimeType": "application/text",
      "encoding": "utf-8"
    }
  ],
  "statistics": {
    "group": "testGroupName",
    "operation": "TI212",
    "category": "categoryName",
    "tags": [
      "tag1",
      "TEST"
    ]
  },
  "subject": "Subject of email",
  "htmlBody": "<b>Html body/b>",
  "scenario": "ScenarioName",
  "testmode": false
}
```

Listing 29: Example for POST request for sending email (simple) with all fields

5.2.2 C#

Unlike example 1, 2 recipients are set here and all parameters of the request are set on the pointer.

```
//Create EmailSendRequest
EmailSendRequest emailRequest = new EmailSendRequest();

//set 2 recipients
emailRequest.To = new List<EmailAddressType>() {
        new EmailAddressType() { AddressName = "NameOfEmail", EmailAddress = "email@email.com"},
        new EmailAddressType() { AddressName = "NameOfEmail2", EmailAddress = "email2@email.com"},

};

var utf8 = new UTF8Encoding();

emailRequest.Attachments = new List<FileHandlerType>(){
    new FileHandlerType()
    {
        Filename = "testFile1",
        DataHandler = utf8.GetBytes("VG90byBqZSB0ZXN0b3ZhY2lhIHByaWxvaGE"),
        ContentID = "54321",
        MimeType = "application/text",
        Encoding = "UTF-8"
    }
};

emailRequest.Statistics = new StatisticsType()
{
    Group = "testGroupName",
    Operation = "TI212",
    Category = "categoryName",
    Tags = new List<string>() { "tag1", "TEST" }
};
emailRequest.Subject = "Subject of email";
emailRequest.HtmlBody = "<b>Html body/b>";
emailRequest.Scenario = "ScenarioName";
emailRequest.Testmode = false;
```

Complete source file is on the next listening 30

```
try
{
    //Create CustomOpenApiClient
    string url = $"{Configuration.DOMAIL_URL}/domail-input-rest/";
    var client = new CustomOpenApiClient(url);

    //Create EmailSendRequest
    EmailSendRequest emailRequest = new EmailSendRequest();

    //set 2 recipients
    emailRequest.To = new List<EmailAddressType>() {
            new EmailAddressType() { AddressName = "NameOfEmail", EmailAddress = "email@email.com"},
            new EmailAddressType() { AddressName = "NameOfEmail2", EmailAddress = "email2@email.com"},

    };

    var utf8 = new UTF8Encoding();
```

```

    emailRequest.Attachments = new List<FileHandlerType>(){
        new FileHandlerType()
        {
            Filename = "testFile1",
            DataHandler = utf8.GetBytes("VG90byBqZSB0ZXN0b3ZhY2lhIHByaWxvaGE"),
            ContentID = "54321",
            MimeType = "application/text",
            Encoding = "UTF-8"
        }
    };

    emailRequest.Statistics = new StatisticsType()
    {
        Group = "testGroupName",
        Operation = "TI212",
        Category = "categoryName",
        Tags = new List<string>() { "tag1", "TEST" }
    };
    emailRequest.Subject = "Subject of email";
    emailRequest.HtmlBody = "<b>Html body/b>";
    emailRequest.Scenario = "ScenarioName";
    emailRequest.Testmode = false;

    //send reqest throug REST API EmailSendSimpleAsync
    Task<CommunicationId> taskEmailSendSimpleAsync = client.EmailSendSimpleAsync(emailRe
     quest);

    //wait for resposne
    var result = taskEmailSendSimpleAsync.GetAwaiter().GetResult();

    if (result.Id > 0)
        Console.WriteLine($"call POST OK - result.Id={result.Id}");

}
catch (ApiException<RestErrorResponse> e1)
{
    Console.WriteLine($"ApiException: {e1}\nStatusCode: {e1.StatusCode}\nResult.Response
     Code: {e1.Result?.ResponseCode}\nResult.Message: {e1.Result?.Message}");
}
catch (ApiException e2)
{
    Console.WriteLine($"ApiException: {e2}, StatusCode: {e2.StatusCode}");
}
catch (Exception ex)
{
    Console.WriteLine($"Exception: {ex}");
}
```

Listing 30: Example for POST request for sending email (simple) only with required fields in C#

5.2.3 Java

In the following code, Java code introduces parameters that enhance the functionality of the email sending service by including statistical categorization, email attachments, and HTML body content. These parameters provide better tracking, organization, and presentation of emails, ensuring a more robust and flexible email communication system. Below is a detailed explanation of the added parameters:
Data Payload

- Statistics
  - group: Indicates the group to which the email belongs.
  - operation: Specifies the type of operation.
  - category: Categorizes the email based on its nature or source.
  - tags: An array of tags for tracking and categorizing the email.
- Attachments
  - filename: The name of the attachment file.
  - dataHandler: The base64 encoded content of the attachment.
  - contentID: A unique identifier for the attachment.
  - mimeType: The MIME type of the attachment.
  - encoding: The encoding used for the attachment content.
- HTML Body
  - htmlBody: Contains the HTML formatted body of the email, which can include rich text formatting such as bold text, images, links, etc.
  - This parameter replaces the plainTextBody parameter, allowing for more visually appealing and formatted email content.

```java
// Define new variables
private static final String GROUP = "your_group";
private static final String OPERATION = "your_operation";
private static final String CATEGORY = "your_category";
private static final String TAG1 = "your_tag1";

private static final String FILENAME = "testFile.txt";
private static final String DATA_HANDLER = "SGVsbG8gY3VzdG9tZXIuDQoNClRoYW5rIHlvdSBmb3
   IgZG93bmxvYWRpbmcgZG9tYWlsLg0KDQpCZXN0IHJlZ2FyZHMNCmRvbWFpbCB0ZWFt";
private static final String CONTENT_ID = "123";
private static final String MIME_TYPE = "application/text";
private static final String ENCODING = "utf-8";

private static final String HTML_BODY = "<b>Html body</b>";

JSONObject statistics = new JSONObject()
  .put("group", GROUP)
  .put("operation", OPERATION)
  .put("category", CATEGORY)
  .put("tags", new JSONArray().put(TAG1)
);

JSONArray arrAttachments = new JSONArray();
arrAttachments.put(new JSONObject()
  .put("filename", FILENAME)
  .put("dataHandler", DATA_HANDLER)
  .put("contentID", CONTENT_ID)
  .put("mimeType", MIME_TYPE)
  .put("encoding", ENCODING)
);

// Create a JSON object with new email details
json.put("htmlBody", HTML_BODY);
json.put("attachments", arrAttachments);
```

```
34  json.put("statistics", statistics);
```

Listing 31: Data Payload

5.2.4 Python

In this section, we demonstrate calling the email sending service with all parameters filled, except for plainTextBody, which is replaced by htmlBody. The code example also includes sending an email attachment and specifying statistics. The statistics field contains four parameters, which are used solely for statistical purposes and categorization of specific emails. These fields help organize and track emails based on internal classifications, aiding in better analysis and management of email operations.
Data Payload

- htmlBody: The HTML content of the email is specified using the htmlBody field.
- scenario: Indicates the scenario to be used.
- subject: The subject of the email.
- testmode: A boolean indicating whether to use test mode.
- to: A list of recipient details, each containing addressName and emailAddress.
- attachments: A list of attachments to be included in the email. Each attachment contains:
  - contentID: A unique identifier for the attachment.
  - dataHandler: The base64 encoded content of the attachment.
  - encoding: The encoding used for the attachment content.
  - filename: The name of the attachment file.
  - mimeType: The MIME type of the attachment.
- statistics: An object specifying statistics for the email, including:
  - group: The group to which the email belongs.
  - operation: The operation type.
  - category: The category of the email.
  - tags: Tags associated with the email for tracking purposes.

```
1   data = {
2     "scenario": "Default_REST",
3     "subject": "Test sending email via Python REST.",
4     "testmode": False,
5     "htmlBody": "<b>Html example body</b>",
6     "to": [
7       {
8         "addressName": "Recipient",
9         "emailAddress": "recipient@example.com"
10      }
11    ],
12    "attachments": [
```

```
    {
      "contentID": "123",
      "dataHandler": "SGVsbG8gY3VzdG9tZXIuDQoNClRoYW5rIHlvdSBmb3IgZG93bmxvYWRpbmcgZG9tYW
     lsLg0KDQpCZXN0IHJlZ2FyZHMNCmRvbWFpbCB0ZWFt",
      "encoding": "utf-8",
      "filename": "testFile.txt",
      "mimeType": "application/text"
    }
  ],
  "statistics": {
    "group": "your_group",
    "operation": "your_operation",
    "category": "your_category",
    "tags": [
        "your_tag1",
        "your_tag2"
    ]
  }
  }
```

Listing 32: Data Payload

5.3 Example 3 - Send email through REST API - EmailSendAdvanced

This example describes how to call EmailSendAdvanced only with required fields.

5.3.1 JSON

Now follows the simple HTTP POST example (33) to call endpoint:
http:\\{yourUrl}/domail-input-rest/email/sendAdvanced

```
{
  "extId": 100,
  "sysId": 200,
  "to": [
    {
          "addressName": "NameOfEmail",
          "emailAddress": "email@email.com"
    }
  ]
}
```

Listing 33: Example for POST request for sending email (advanced) only with required fields

5.3.2 C#

Complete source file is on the next listening 34

```
try
{
    //Create CustomOpenApiClient
    string url = $"{Configuration.DOMAIL_URL}/domail-input-rest/";
    var client = new CustomOpenApiClient(url);

    //Create EmailSendAdvancedRequest
    var emailRequest = new EmailSendAdvancedRequest();

    //set 1 recipients
```

```
    emailRequest.To = new List<EmailAddressType>() {
            new EmailAddressType() { AddressName = "NameOfEmail", EmailAddress = "email@email.com"},
    };
    emailRequest.ExtId = "100";
    emailRequest.SysId = "200";

    //send reqest throug REST API EmailSendSimpleAsync
    Task<CommunicationId> taskEmailSendSimpleAsync = client.EmailSendAdvancedAsync(emailRequest);

    //wait for resposne
    var result = taskEmailSendSimpleAsync.GetAwaiter().GetResult();

    if (result.Id > 0)
        Console.WriteLine($"call POST OK - result.Id={result.Id}");

}
catch (ApiException<RestErrorResponse> e1)
{
    Console.WriteLine($"ApiException: {e1}\nStatusCode: {e1.StatusCode}\nResult.ResponseCode: {e1.Result?.ResponseCode}\nResult.Message: {e1.Result?.Message}");
}
catch (ApiException e2)
{
    Console.WriteLine($"ApiException: {e2}, StatusCode: {e2.StatusCode}");
}
catch (Exception ex)
{
    Console.WriteLine($"Exception: {ex}");
}
```

Listing 34: Example for POST request for sending email (advanced) only with required fields in C#

5.3.3 Java

Another language option is Java. This enhanced Java program illustrates how to send an email by utilizing the HttpURLConnection class to communicate with a REST API. The program builds a JSON request body containing email details and dispatches a POST request to the designated URL, employing Basic Authentication for API access.

5.3.3.1 Imports

- InputStream and OutputStream: For reading and writing data streams.
- HttpURLConnection: For HTTP communication.
- URL: Represents a URL object.
- Base64: For encoding credentials.
- Scanner: For reading input streams.
- JSONArray and JSONObject: For constructing JSON objects and arrays.

```
import java.io.InputStream;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.nio.charset.StandardCharsets;
import java.util.Base64;
import java.util.Scanner;

import org.json.JSONArray;
import org.json.JSONObject;
```

Listing 35: Import libraries

5.3.3.2 Main Class and Method

- DOMAIL_URL: The endpoint URL for the email sending service. Replace DOMAIL_URL with the actual URL.
- USERNAME and PASSWORD: Credentials for Basic Authentication.

```
public class Main {

  private static final String DOMAIL_URL = "{{DOMAIL_URL}}/domail-input-rest/email/sen
   dAdvanced";
  private static final String USERNAME = "username";
  private static final String PASSWORD = "XXXXXXXXXX";

  public static void main(String[] args) {
      try {
          // Construct the JSON payload
          JSONObject jsonPayload = createJsonPayload();

          // Send the POST request
          HttpURLConnection connection = sendPostRequest(jsonPayload);

          // Handle the response
          handleResponse(connection);
      } catch (Exception e) {
          e.printStackTrace();
      }
  }
```

Listing 36: Main Class and Method

5.3.3.3 Create JSON Payload

- createJsonPayload: Constructs the JSON object representing the request body with the following parameters:
  - scenario: The scenario for the email (e.g., "Default_REST").
  - to: A list of recipients.
  - extId: An external identifier for tracking.
  - sysId: An identifier of system.

```
  private static JSONObject createJsonPayload() {
    JSONArray recipients = new JSONArray();
    recipients.put(new JSONObject().put("addressName", "Recipient").put("emailAddress",
     "recipient@example.com"));

    JSONObject jsonPayload = new JSONObject();
    jsonPayload.put("scenario", "Default_REST");
    jsonPayload.put("to", recipients);
    jsonPayload.put("extId", "2600");
    jsonPayload.put("sysId", "100");

    return jsonPayload;
}
```

Listing 37: Create JSON Payload

5.3.3.4 Send POST Request

- sendPostRequest: Opens a connection, sets up the request, and sends the JSON payload.
- Basic Authentication: Encodes the username and password, setting it in the request header.

```
  private static HttpURLConnection sendPostRequest(JSONObject jsonPayload) throws Except
     ion {
    URL url = new URL(DOMAIL_URL);
    HttpURLConnection connection = (HttpURLConnection) url.openConnection();

    connection.setRequestMethod("POST");
    connection.setRequestProperty("Content-Type", "application/json; utf-8");
    connection.setRequestProperty("Accept", "application/json");
    connection.setDoOutput(true);

    String userCredentials = USERNAME + ":" + PASSWORD;
    String basicAuth = "Basic " + Base64.getEncoder().encodeToString(userCredentials.get
     Bytes(StandardCharsets.UTF_8));
    connection.setRequestProperty("Authorization", basicAuth);

    try (OutputStream os = connection.getOutputStream()) {
        byte[] input = jsonPayload.toString().getBytes(StandardCharsets.UTF_8);
        os.write(input, 0, input.length);
    }

    return connection;
}
```

Listing 38: Send POST Request

5.3.3.5 Handle Response

- handleResponse: Retrieves and prints the response code and message, reads and prints the response content.

```
  private static void handleResponse(HttpURLConnection connection) throws Exception {
    int responseCode = connection.getResponseCode();
```

```
    System.out.println("Response Code: " + responseCode + " " + connection.getResponseMe
     ssage());

    InputStream responseStream = connection.getErrorStream();
    if (responseStream == null) {
        responseStream = connection.getInputStream();
    }

    try (Scanner scanner = new Scanner(responseStream)) {
        scanner.useDelimiter("\\Z");
        System.out.println(scanner.next());
    }
}
}
```

Listing 39: Handle Response

5.3.4 Python

This Python script demonstrates how to send an email using the domail-input-rest service with the sendAdvanced endpoint. It uses the requests library for making HTTP requests and Basic Authentication for securing the API call.

5.3.4.1 Imports

- requests: A library for making HTTP requests.
- HTTPBasicAuth: A class for handling HTTP Basic Authentication.

```
import requests
from requests.auth import HTTPBasicAuth
```

Listing 40: Import libraries

5.3.4.2 Configuration

- url: The endpoint URL for the email sending service. Replace DOMAIL_URL with your URL.
- user: Your username for Basic Authentication.
- password: Your password for Basic Authentication. Replace XXXXXXXXXX with your actual password.

```
url = '{{DOMAIL_URL}}/domail-input-rest/email/sendAdvanced'

user = 'username'
password = 'XXXXXXXXXX'
```

Listing 41: Configuration

5.3.4.3 Data Payload

- scenario: The scenario for the email, e.g., "Default_REST".
- sysId: An identifier of system.
- extId: An external identifier.
- to: A list of recipients. Each recipient includes:
  - addressName: The name of the recipient.
  - emailAddress: The email address of the recipient.

```
data = {
  "scenario": "Default_REST",
  "sysId": "100",
  "extId": "200",
  "to": [
    {
      "addressName": "Recipient",
      "emailAddress": "recipient@example.com"
    }
  ],
}
```

Listing 42: Data Payload

5.3.4.4 Sending the Request

- requests.post: Sends a POST request to the specified URL with the provided JSON data and authentication.
- response.raise_for_status(): Checks if the request was successful. If not, it raises an HTTPError.
- except requests.exceptions.RequestException as e: Catches any exceptions that occur during the request and prints an error message.

```
try:
  # Send a POST request with the JSON data and Basic Authentication
  response = requests.post(url, json=data, auth=HTTPBasicAuth(user, password))

  # Check if the request was successful
  response.raise_for_status()

  # Print the response status code and content
  print(f"Response Code: {response.status_code}")
  print("Response Content:", response.json())

except requests.exceptions.RequestException as e:
  # Print the error if the request fails
  print(f"An error occurred: {e}")
```

Listing 43: Sending the Request

5.4 Example 4 - Send email through REST API - EmailSendAdvanced

This example describes how to call EmailSendAdvanced with all fields.

5.4.1 JSON

In the next example (44), there is POST request with all fields:

```
{
  "to": [
    {
      "addressName": "NameOfEmail",
      "emailAddress": "email@email.com"
    },
    {
      "addressName": "NameOfEmail2",
      "emailAddress": "email2@email.com"
    }
  ],
  "subject": "Subject of email",
  "htmlBody": "<b>Html body</b>",
  "scenario": "ScenarioName",
  "attachments": [
    {
      "filename": "testFile1",
      "dataHandler": "VG90byBqZSB0ZXN0b3ZhY2lhIHByaWxvaGE=",
      "contentID": "54321",
      "mimeType": "application/text",
      "encoding": "utf-8"
    }
  ],
  "statistics": {
    "group": "testGroupName",
    "operation": "TI212",
    "category": "categoryName",
    "tags": ["tag1", "TEST", "tag2"]
  },
  "testmode": false,
  "extId": "100",
  "sysId": "200",
  "advancedRequestData": {
    "priority": false,
    "duplicity": false
  }
}
```

Listing 44: Example for POST request for sending email (simple) with all fields

5.4.2 C#

Complete source file is on the next listening 45

```
try
{
    string url = $"{Configuration.DOMAIL_URL}/domail-input-rest/";
    var client = new CustomOpenApiClient(url);//Create CustomOpenApiClient

    //Create EmailSendAdvancedRequest
    var emailRequest = new EmailSendAdvancedRequest();
    //set 2 recipients
    emailRequest.To = new List<EmailAddressType>() {
            new EmailAddressType() { AddressName = "NameOfEmail", EmailAddress = "email@email.com"},
            new EmailAddressType() { AddressName = "NameOfEmail2", EmailAddress = "email2@email.com"},
    };
```

```

    var utf8 = new UTF8Encoding();
    emailRequest.Attachments = new List<FileHandlerType>(){
        new FileHandlerType()
        {
            Filename = "testFile1",
            DataHandler = utf8.GetBytes("VG90byBqZSB0ZXN0b3ZhY2lhIHByaWxvaGE="),
            ContentID = "54321",
            MimeType = "application/text",
            Encoding = "UTF-8"
        }
    };
    emailRequest.Statistics = new StatisticsType()
    {
        Group = "testGroupName",
        Operation = "TI212",
        Category = "categoryName",
        Tags = new List<string>() { "tag1", "TEST", "tag2" }
    };
    emailRequest.Subject = "Subject of email";
    emailRequest.HtmlBody = "<b>Html body/b>";
    emailRequest.Scenario = "ScenarioName";
    emailRequest.Testmode = false;
    emailRequest.ExtId = "100";
    emailRequest.SysId = "200";
    emailRequest.AdvancedRequestData = new AdvancedRequestType()
    {
        Priority = false,
        Duplicity = false
    };
    //send reqest throug REST API EmailSendSimpleAsync
    Task<CommunicationId> taskEmailSendSimpleAsync = client.EmailSendAdvancedAsync(email
     Request);

    //wait for resposne
    var result = taskEmailSendSimpleAsync.GetAwaiter().GetResult();

    if (result.Id > 0)
        Console.WriteLine($"call POST OK - result.Id={result.Id}");
}
catch (ApiException<RestErrorResponse> e1)
{
    Console.WriteLine($"ApiException: {e1}\nStatusCode: {e1.StatusCode}\nResult.Response
     Code: {e1.Result?.ResponseCode}\nResult.Message: {e1.Result?.Message}");
}
catch (ApiException e2)
{
    Console.WriteLine($"ApiException: {e2}, StatusCode: {e2.StatusCode}");
}
catch (Exception ex)
{
    Console.WriteLine($"Exception: {ex}");
}
```

Listing 45: Example for POST request for sending email (advanced) with all fields in C#

5.4.3 Java

This Java program demonstrates how to send an email using the HttpURLConnection class to interact with a REST API. It constructs a JSON request body with

detailed email attributes and sends a POST request to the specified URL. Basic Authentication is utilized for API access. Users can send either plain text or HTML emails, depending on their needs.

5.4.3.1 Key Fields in the JSON Payload

- scenario: Defines the scenario under which the email is being sent.
- extId and sysId: External and system identifiers for tracking purposes.
- to: An array of recipient objects, each containing the recipient's name and email address.
- statistics: Provides categorization for statistical purposes.
- attachments: An array of attachment objects, each with details about the file.
- testmode: Indicates whether the email is sent in test mode.
- subject: The subject line of the email.
- plainTextBody or htmlBody: The body of the email, which can be either plain text or HTML.
- additionalContent: Additional content such as QR codes.
- advancedRequestData: Contains advanced request parameters like duplicity, priority, and scheduled send time.

```
// required attributes
json.put("scenario", "Default_REST");
json.put("extId", "2600");
json.put("sysId", "100");
JSONArray arrTo = new JSONArray();
arrTo.put(new JSONObject().put("addressName", "Recipient").put("emailAddress", "recipi
   ent@example.com"));
json.put("to", arrTo);

//optional attributes
JSONObject statistics = new JSONObject()
    .put("group", "Test")
    .put("operation", "REST")
    .put("category", "Java")
    .put("tags", new JSONArray().put("domail"));
json.put("statistics", statistics);
JSONArray arrAttachments = new JSONArray();
arrAttachments.put(new JSONObject()
    .put("filename", "testFile1.txt")
    .put("dataHandler", "VG90byBqZSB0ZXN0b3ZhY2lhIHByaWxvaGE=")
    .put("contentID", "54321")
    .put("mimeType", "application/text")
    .put("encoding", "utf-8"));
json.put("attachments", arrAttachments);
json.put("testmode", false);
json.put("subject", "Test sending email via Java REST.");

// json.put("plainTextBody", "Hello, this is test of sending email via Java.");
json.put("htmlBody", "<b>Html body</b>");

JSONArray arrAdditionalContent = new JSONArray();
```

```
arrAdditionalContent.put(new JSONObject()
    .put("key", "QRCODE")
    .put("value", "aHR0cHM6Ly9kb21haWwuaW8vaG9tZQ==", ));
json.put("additionalContent", arrAdditionalContent);
JSONObject advancedRequestData = new JSONObject()
    .put("duplicity", true)
    .put("priority", false)
    .put("sendTime", "2024-07-20T10:53:40");
json.put("advancedRequestData", advancedRequestData);
```

Listing 46: Key Fields in the JSON Payload

5.4.4 Python

This Python script illustrates how to send an email with all advanced attributes using the domail-input-rest service at the sendAdvanced endpoint. It employs the requests library for making HTTP requests and uses Basic Authentication to secure the API call.

5.4.4.1 Imports

- requests: A library for making HTTP requests.
- HTTPBasicAuth: A class for handling HTTP Basic Authentication.

```
import requests
from requests.auth import HTTPBasicAuth
```

Listing 47: Import libraries

5.4.4.2 Configuration

- url: The endpoint URL for the email sending service. Replace DOMAIL_URL with your URL.
- user: Your username for Basic Authentication.
- password: Your password for Basic Authentication. Replace XXXXXXXXXX with your actual password.

```
url = '{{DOMAIL_URL}}/domail-input-rest/email/sendAdvanced'

user = 'username'
password = 'XXXXXXXXXX'
```

Listing 48: Configuration

5.4.4.3 Data Payload

- scenario: The scenario for the email, e.g., "Default_REST".
- sysId: An identifier of system.
- extId: An external identifier.

- to: A list of recipients. Each recipient includes:
  - addressName: The name of the recipient.
  - emailAddress: The email address of the recipient.
- subject: The subject of the email.
- htmlBody: The HTML content of the email. You can use the field "plainTextBody" instead.
- testmode: Flag indicating if the email is sent in test mode.
- statistics: A statistical categorization fields:
  - group: Group categorization for statistics.
  - operation: Operation type for statistics.
  - category: Category of statistics.
  - tags: Tags for further categorization.
- advancedRequestData: A dictionary containing advanced request parameters:
  - duplicity: Flag for handling duplicate emails.
  - priority: Flag indicating if the email should be sent with priority.
  - sendTime: Scheduled time for sending the email.
- additionalContent: A list of dictionaries for additional content:
  - key: Key for additional content.
  - value: Base64 encoded value for additional content.
- attachments: A list of attachments to be included in the email. Each attachment contains:
  - contentID: A unique identifier for the attachment.
  - dataHandler: The base64 encoded content of the attachment.
  - encoding: The encoding used for the attachment content.
  - filename: The name of the attachment file.
  - mimeType: The MIME type of the attachment.

```
data = {
  # "plainTextBody": "Hello, this is test of sending email via Python.",
  "scenario": "Default_REST",
  "subject": "Test sending email via Python REST.",  # The subject of the email
  "htmlBody": "<b>Html body</b>",  # The HTML content of the email
  "testmode": False,  # Flag indicating if the email is sent in test mode
  "statistics": {
      "group": "Test",  # Group categorization for statistics
      "operation": "REST",  # Operation type for statistics
      "category": "Python",  # Category of statistics
      "tags": [
          "domail",
          "advanced"
      ]  # Tags for further categorization
  },
```

```
    "advancedRequestData": {
        "duplicity": True,  # Flag for handling duplicate emails
        "priority": True,  # Flag indicating if the email should be sent with priority
        "sendTime": "2024-07-15T12:19:40"  # Scheduled time for sending the email
    },
    "additionalContent": [{
        "key": "QRCODE",  # Additional content key
        "value": "aHR0cHM6Ly9kb21haWwuaW8vaG9tZQ==”  # Base64 encoded value for addition
     al content
    }],
    "attachments": [
        {
            "contentID": "54321",  # Identifier for the attachment
            "dataHandler": "VG90byBqZSB0ZXN0b3ZhY2lhIHByaWxvaGE=",  # Base64 encoded dat
     a of the attachment
            "encoding": "utf-8",  # Encoding of the attachment
            "filename": "testFile1.txt",  # Name of the attachment file
            "mimeType": "application/text"  # MIME type of the attachment
        }
    ],
    "sysId": "100",
    "extId": "200",
    "to": [
        {
            "addressName": "Recipient",
            "emailAddress": "example@example.com"
        }
    ],
}
```

Listing 49: Data Payload

5.4.4.4 Sending the Request

- requests.post: Sends a POST request to the specified URL with the provided JSON data and authentication.
- response.raise_for_status(): Checks if the request was successful. If not, it raises an HTTPError.
- except requests.exceptions.RequestException as e: Catches any exceptions that occur during the request and prints an error message.

```
try:
  # Send a POST request with the JSON data and Basic Authentication
  response = requests.post(url, json=data, auth=HTTPBasicAuth(user, password))

  # Check if the request was successful
  response.raise_for_status()

  # Print the response status code and content
  print(f"Response Code: {response.status_code}")
  print("Response Content:", response.json())

except requests.exceptions.RequestException as e:
  # Print the error if the request fails
  print(f"An error occurred: {e}")
```

Listing 50: Sending the Request

5.5 Reference guide

Name Description	Script
Attachment is Compare whether attachment (name, type) is same or contains (using * and ? characters) specified text	attachment == "*.png"
Content of email is Compare whether content of email is same or contains (using * and ? characters) specified text	content == "*@domail.eu"
Content request is Compare whether content of request is same or contains (using * and ? characters) specified text	content == "*@domail.eu"
Count attachments Compare count of attachments	attachment.count >= 0
Count "BCC" recipients Compare count of recipients "BCC"	bcc.count > 0
Count "CC" recipients Compare count of recipients "CC"	cc.count > 0
Count of attachments Compare count of attachments	attachment.count >= 0
Count "TO" recipients Compare count of recipients "TO"	to.count > 0
Domain "TO" is Compare whether domain of recipient "TO" is same or contains (using * and ? characters) specified text	to.domain == "domail.eu"
Field "category" is Compare whether field "category" is same or contains (using * and ? characters) specified text	category == "domail.eu"
Field "duplicity" is Compare whether field "duplicity" has same logical value	duplicity == true
Field "extId" is Compare whether field "extId" is same or contains (using * and ? characters) specified text	extId == "domail.eu"
Field "group" is Compare whether field "group" is same or contains (using * and ? characters) specified text	group == "domail.eu"
Field "instance" is	instance == "domail.eu"

Name **Description**	**Script**
Compare whether field "instance" is same or contains (using * and ? characters) specified text	
Field "operation" is Compare whether field "operation" is same or contains (using * and ? characters) specified text	operation == "domail.eu"
Field "priority" is Compare whether field "priority" has same logical value	priority == true
Field "sysId" is Compare whether field "sysId" is same or contains (using * and ? characters) specified text	sysId == "domail.eu"
Field "tags" is Compare whether field "tags" is same or contains (using * and ? characters) specified text	tags == "domail.eu"
Field "testMode" is Compare whether field "testMode" has same logical value	testMode == true
Filter attachments Filter attachments by requested attributes and evaluate condition	attachment.filename("*.png") .contentType("image/*").count >= 1
Filter attachments Filter attachments by requested attributes and evaluates condition	attachment.filename("*.png") .contentType("image/*").count >= 1
Logical operator "and" Condition is true only if left and right side of condition are true	and
Logical operator "and" Condition is true only if left and right side of condition is true	and
Logical operator "negation" Negate condition in braces - true <-> false	!()
Logical operator "or" Condition is true if left or right side of condition is true	or
Logical operator "or" Condition true is if left or right side of condition is true	or
Logical operator "or" Condition true is if true is left or right side of condition	or
Recipient "BCC" is Compare whether recipient "BCC" is same or contains (using * and ? characters) specified text	bcc == "*@domail.eu"

Name **Description**	**Script**
Recipient "CC" is Compare whether recipient "CC" is same or contains (using * and ? characters) specified text	cc == "*@domail.eu"
Recipient "TO" is Compare whether recipient "TO" is same or contains (using * and ? characters) specified text	to == "*@domail.eu"
Sender is Compare whether sender is same or contains (using * and ? characters) specified text	from == "*@domail.eu"
Size of attachments Compare size of attachments	attachment.size >= 1MB
Size of email is Compare total size of email	size >= 1MB
Size of request is Compare total size of request	size >= 1MB
Subject is Compare whether subject is same or contains (using * and ? characters) specified text	subject == "*@domail.eu"
Type of attachments is Compare whether type of attachment (mime type) is same or contains (using * and ? characters) specified text	attachment.contentType == "image/*"

5.6 Scenarios selection scripts (conditions)

Scenario selection scripts enable dynamic decisions on which scenario to apply based on the content and parameters of an incoming SMTP message. These scripts leverage Spring Expression Language (SpEL), allowing flexibility in defining conditions in a Java-like scripting language. They act as filters that analyze email parameters, such as the number and size of attachments, to decide whether a scenario is applied. Scripts must return a `true` or `false` value to determine the selection outcome. Setting conditions is managed in the scenario details section of an SMTP-type scenario, under the "Use conditions" tab.

- **Spring Expression Language (SpEL)**
  SpEL provides a straightforward way to construct expressions to evaluate email parameters. It supports a wide range of comparison operations, access to object fields, string and logical operations, and mathematical expressions. You can refer to the SpEL documentation here.

5.6.1 Compose conditions for scenario selection

Conditions for scenario selection are divided into basic and advanced categories, each allowing different levels of filtering based on SMTP message parameters.

5.6.1.1 Basic conditions

Basic conditions provide simple comparisons and checks of key message parameters, such as sender, recipient, subject, and message size. Common basic conditions include:

- **Comparison with specific values**
  - For example, checking if the sender or recipient matches specific text:
    from == "*@domail.io"
- **Count and size of attachments**
  - Determining if a message has a required number of recipients or attachments:
    to.count > 0
    attachment.count >= 1
- **Checking MIME type and attachment name**
  - Simple check for the type or name of attachment files with
    attachment.contentType == "image/*"

5.6.1.2 Advanced conditions

Advanced conditions combine multiple basic conditions using logical operators such as `and`, `or`, and negation (`!`). These conditions offer more sophisticated filtering based on multiple criteria simultaneously:

- **Combined checks**
  - Combining multiple conditions with and to ensure multiple criteria are met simultaneously:
    from == "*@domail.io" and size >= 1MB
- **Negating conditions**
  - Negating conditions to exclude certain messages, such as blocking processing for certain types of attachments or specific recipients:
    !(subject == "test")
- **Filtering attachments with multiple criteria**
  - Filtering attachments based on multiple properties, such as filename, type, and count, for example:
    attachment.filename("*.png").contentType("image/*").count >= 1

This flexibility enables developers to precisely define rules for scenario selection based on a variety of SMTP message parameters, allowing the application to meet specific email processing requirements.

5.6.1.3 List of conditions

Name Description	Script
Sender is Compare whether sender is same or contains (using * and ? characters) specified text.	from == "*@domail.eu"
Subject is Compare whether subject is same or contains (using * and ? characters) specified text.	subject == "*@domail.eu"
Recipient "TO" is Compare whether recipient "TO" is same or contains (using * and ? characters) specified text.	to == "*@domail.eu"
Domain "TO" is Compare whether domain of recipient "TO" is same or contains (using * and ? characters) specified text.	to.domain == "domail.eu"
Count "TO" recipients Compare count of recipients "TO".	to.count > 0
Recipient "CC" is Compare whether recipient "CC" is same or contains (using * and ? characters) specified text.	cc == "*@domail.eu"
Count "CC" recipients Compare count of recipients "CC".	cc.count > 0
Recipient "BCC" is Compare whether recipient "BCC" is same or contains (using * and ? characters) specified text.	bcc == "*@domail.eu"
Count "BCC" recipients Compare count of recipients "BCC".	bcc.count > 0
Content of email is Compare whether content of email is same or contains (using * and ? characters) specified text.	content == "*@domail.eu"
Size of email is Compare total size of email.	size >= 1MB
Attachment is Compare whether attachment (name, type) is same or contains (using * and ? characters) specified text.	attachment == "*.png"
Type of attachments is Compare whether type of attachment (mime type) is same or contains (using * and ? characters) specified text.	attachment.contentType == "image/*"

Name Description	Script
Count of attachments Compare count of attachments.	attachment.count >= 0
Size of attachments Compare size of attachments.	attachment.size >= 1MB
Filter attachments Filter attachments by requested attributes and evaluate condition.	attachment.filename("*.png").contentType ("image/*").count >= 1
Logical operator "and" Condition is true only if left and right side of condition are true.	and
Logical operator "or" Condition is true if left or right side of condition is true.	or
Logical operator "negation" Negate condition in braces - true <-> false.	!()

5.7 Processing scripts

5.7.1 ECMA/Javascript

ECMAScript [1] is a standard for scripting languages, including JavaScript, JScript, and ActionScript. It is also best known as a JavaScript standard intended to ensure the interoperability of web pages across different web browsers.It is standardized by Ecma International in the document ECMA-262.

ECMAScript is commonly used for client-side scripting on the World Wide Web (WWW), and it is increasingly being used to write server-side applications and services using Node.js and other runtime environments.
In the following ECMA/Javascript code block 51, there is an example of a default script, which you can of course set yourself according to your requirements.

```
to();           // set address "TO" from request
text();         // set text email from request
html();         // set html text emailu from request
subject();      // set subject from request
from();         // set address "FROM" from request
returnPath();   // set returnPath from request
replyTo();      // set replyTo from request

smtpServer();   // set SMTP server
attachment();   // set attachments from request
trackPixel();   // set track pixel, if email is in HTML mode
```

Listing 51: Example of default ECMA/Javascript for processing email

5.7.2 Reference guide for processing scripts

5.7.2.1 Work with attachments

Script / Description
attachment.add("filename"); Add attachment from file system. Directory with attachments is specified by parameter of scenario
attachment.add("filename","name"); Add attachment from file system. Name of attachment in email may be different. Directory with attachments is specified by parameter of scenario.
attachment.addFromGallery("filename"); Add attachment from gallery.
attachment.addFromGallery("filename","name"); Add attachment from gallery. Name of attachment in email may be different.
attachment.add(index); Add attachment from request on specified index

5.7.2.2 Work with 'Bcc' addresses

Script / Description
bcc(); set 'bcc' adress(es) from settings of scenario
bcc.add("address"); add single address of type Bcc
bcc.add("address1", "address2", ...); add multiple addresses of type Bcc
bcc.addEmailWithName("email", "name"); add single address with name of type Bcc
bcc.clear(); remove all addresses of type Bcc
bcc.count(); return count of addresses set of type Bcc
bcc.set("address"); set single address of type Bcc
bcc.set("address1", "address2", ...); set multiple addresses of type Bcc
bcc.setEmailWithName("email", "name"); set single address with name of type Bcc

5.7.2.3 Work with 'Cc' addresses

Script / Description
cc(); set 'cc' adress(es) from settings of scenario
cc.add("address"); add single address of type Cc
cc.add("address1", "address2", ...); add multiple addresses of type Cc
cc.addEmailWithName("email", "name"); add single address with name of type Cc
cc.clear(); remove all addresses of type Cc
cc.count(); return count of addresses set of type Cc
cc.set("address"); set single address of type Cc
cc.set("address1", "address2", ...); set multiple addresses of type Cc
cc.setEmailWithName("email", "name"); set single address with name of type Cc

5.7.2.4 Context - variables for processing message

Script / Description
ctx.attachments; get List<Attachment<?» object - list of attachments that will be added to email after processing ends
ctx.fail("Error message"); Mark processing as invalid with given description
ctx.getAttachmentsByExtension(".pdf"); get List<Attachment<?» object - list of attachments that will be added to email after processing ends that have specified extension in their name
ctx.getDbEmail(); get srv_emails object - database entry of single email
ctx.getDbReq();

get srv_emlreq object - database entry of request
ctx.getDbReqRun(); get srv_procrun object - database entry of run
ctx.id; get IdNumRunid object - id of actual processing composed of id of request, num - index of email and runid - id of run
ctx.isFailed(); returns true if error ocurred during processing, otherwise return false
ctx.message; get MailMessage object for work with generated message
ctx.message.getMessage(); get SMTPMessage object - generated email
ctx.params; get Map<String, Object> object - map of parameters set during processing
ctx.req; get CreateCommunicationRequestI object - request to send email
ctx.scenarioParams; get Map<String, String> object - map of parameters set for actual scenario
ctx.scriptConstants; get Map<String, String> object - map of set constants
ctx.service; get services to work with
ctx.service.db; get DbService object - service for work with database
ctx.service.fs; get FsService object - service for work with file system

5.7.2.5 Sign email with DKIM

Script / Description
dkim(); if parameter of scenario is set, sign email with DKIM (while sending)
dkim.use(); Sign email with DKIM. Parameter of scenario specifies alias of certificate

5.7.2.6 Work with 'From' addresses

Script / Description
from(); set 'from' address from settings of scenario
from.clear(); remove all addresses of type From
from.count(); return count of addresses set of type From
from.set("address"); set single address of type From
from.setEmailWithName("email", "name"); set single address with name of type From

5.7.2.7 Work with text/html content of message

Script / Description
html(); use HTML content from request
html.set("html"); set specified HTML content of email

5.7.2.8 Logging to database and similar

Script / Description
log.logError("message"); log ERROR message
log.logInfo("message"); log INFO message
log.log("level","message"); log message with given level

5.7.2.9 Set priority

Script / Description
priority();

Set message as priority/nonpriority, based on settings of scenario. If scenario does not have settings keep original value

priority.set();
Set message as priority - send it to priority queue

priority.set(true/false);
Set message as priority/nonpriority

5.7.2.10 Work with 'qr' code

Script / Description
qr(); Set qr code - image in html part of email. QR code is read from iContent part of request. If does not contain html or html does not contain qr section then do nothing
qr.addToEndHtmlPart(); Set qr code - image in html part of email. QR code is read from iContent part of request. If does not contain html then do nothing. If html does not contain qr section then it is added at the end
qr.addToEndHtmlPart("QR code value"); Set given qr code - image in html part of email. If does not contain html then do nothing. If html does not contain qr section then it is added at the end
qr.generate(); Set qr code - image in html part of email. QR code is read from iContent part of request. Email must contain html and html must contain qr section, otherwise it is error
qr.generate("QR code value"); Set given qr code - image in html part of email. Email must contain html and html must contain qr section, otherwise it is error
qr.update("QR code value"); Set given qr code - image in html part of email. If does not contain html or html does not contain qr section then do nothing

5.7.2.11 Work with 'Reply to' addresses

Script / Description
replyTo.add("address"); add single address of type ReplyTo
replyTo.add("address1", "address2", ...);

add multiple addresses of type ReplyTo

replyTo.addEmailWithName("email", "name");
add single address with name of type ReplyTo

replyTo.clear();
remove all addresses of type ReplyTo

replyTo.count();
return count of addresses set of type ReplyTo

replyTo.set("address");
set single address of type ReplyTo

replyTo.set("address1", "address2", ...);
set multiple addresses of type ReplyTo

replyTo.setEmailWithName("email", "name");
set single address with name of type ReplyTo

5.7.2.12 Monitor undelivered message

Script / Description
reportUndelivered.sendEmail(); Prepare email with report of undelivered emails to sending.

5.7.2.13 Work with 'returnPath'

Script / Description
returnPath(); set 'return-path' address from settings of scenario
returnPath.set("address"); set given 'return-path' address

5.7.2.14 Sign email by certificate

Script / Description
signEmail(); Sign email. Sign by parameters of scenario. Attachments must already be added by attachment(Work with attachments) plugin!

5.7.2.15 Sign PDF attachments

Script / Description
signPdf(); Sign all PDF attachments. Sign by parameters of scenario. Attachments must be already added by attachments plugin!

5.7.2.16 Set SMTP server for sending email

Script / Description
smtpServer(); use SMTP server from settings of scenario
smtpServer.use("smtpServerId"); set id of SMTP server for sending email

5.7.2.17 Perform standard processing based on request and scenario settings

Script / Description
standard(); perform standard processing on plugins, f.e. to(); subject();
standard.except(module1, module2...); perform standard processing on all plugins except named plugins
standard.standard(module1, module2...); perform standard processing on named plugins

5.7.2.18 Set subject

Script / Description
subject(); Set subject of email from request
subject.set("Subject of email"); Set subject email

5.7.2.19 Work with text/plain content of message

Script / Description
text(); set text content of email from request
text.set("Text of email"); set text content of email

5.7.2.20 Work with 'To' addresses

Script / Description
to(); set 'to' adress(es) from request and from settings of scenario or from request
to.add("address"); add single address of type To
to.add("address1", "address2", ...); add multiple addresses of type To
to.addEmailWithName("email", "name"); add single address with name of type To
to.clear(); remove all addresses of type To
to.count(); return count of addresses set of type To
to.set("address"); set single address of type To
to.set("address1", "address2", ...); set multiple addresses of type To
to.setEmailWithName("email", "name"); set single address with name of type To

5.7.2.21 Work with 'track pixel'

Script / Description
trackPixel.add();

Set track-pixel image in html part of email. If does not contain html then do nothing. If html does not contain track pixel section then it is added at the end
trackPixel.update();
Set track-pixel image in html part of email. If does not contain html or html does not contain track pixel section then do nothing

5.7.2.22 Validation of message (inputs, outputs)

Script / Description
validate.message();
Validate content of email

5.7.3 To, CC, BCC, Subject

In the following ECMA/Javascript code block 52, we can modify the script that can be used in some scenario. The goal of the script is to modify or add some addresses to the "TO", "CC" or "BCC" fields.

```
to();           // set address "TO" from request
text();         // set text email from request
html();         // set html text emailu from request
subject();      // set subject from request
from();         // set address "FROM" from request
returnPath();   // set returnPath from request
replyTo();      // set replyTo from request

smtpServer();   // set SMTP server
attachment();   // set attachments from request
trackPixel();   // set track pixel, if email is in HTML mode

to.add("example-to@dominanz.sk"); // add fixed address to "TO"

cc();           // set address "CC" from request
cc.add("example-cc@dominanz.sk", "example-cc2@dominanz.sk"); // add fixed addresses to "
    CC"

bcc();          // set address "BCC" from request
bcc.add("example-bcc@dominanz.sk");   // add fixed address to "BCC"
```

Listing 52: ECMA/Javascript for changing fields TO, CC, BCC

This example can be used if we want every communication that will be sent using this script to have a fixed address added, to which we also want to send an email.

On the following figure 16, you can see after processing the email communication how the TO, CC, BCC defined in the script have been added to the EML source.

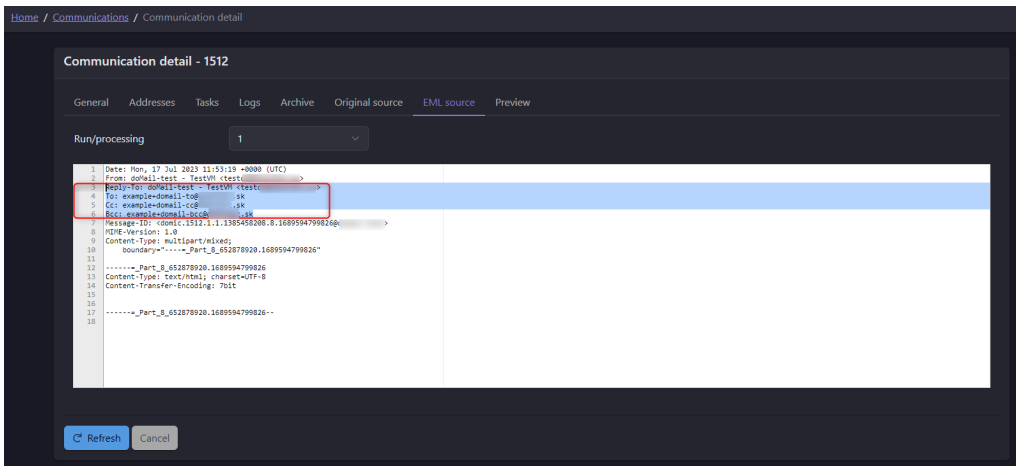

Figure 16: Communication detail - EML source - ExampleEcmaScriptToCcBcc

Another possible way (listing 53) of use can be , if we use the server for example in DEV or UAT mode, so we can overwrite all addresses and set our desired address.

```
...
to.clear();     // clear address "TO" from request
cc.clear();     // clear address "CC" from request
bcc.clear();    // clear address "BCC" from request
to.set("example-to-TEST@dominanz.sk");  // SET fixed address to "TO"
...
```

Listing 53: ECMA/Javascript for changing field TO in test mode

5.7.4 PlainTextBody, HtmlTextBody

In the following example (listing 54) we will show how we can add e.g. a signature to each sending communication in plaing text.

```
...
var plainText = ctx.req.getPlainText();
if(plainText != null && plainText != ""){
    plainText = plainText + '\n\r Pridany podpis na koniec';
    text.set(plainText);
}
...
```

Listing 54: ECMA/Javascript for adding text to body - PLAIN

Result for this request JSON (listing 55):

```
{
    "to": [
        {
            "addressName": "Lukas",
            "emailAddress": "lukas@example.sk"
        }
    ],
    "plainTextBody": "Povodny text emailu",
    "testmode": false
}
```

Listing 55: Example for POST request for sending email with adding plain text in script

Result EML:

```
Date: Thu, 20 Jul 2023 06:28:27 +0000 (UTC)
From: doMail-test - TestVM <test[illegible]>
Reply-To: doMail-test - TestVM <test[illegible]>
To: Lukas [illegible] <lukas [illegible].sk>
Message-ID: <domic.1559.1.1.570591244.57.1689834507143[illegible]>
MIME-Version: 1.0
Content-Type: multipart/mixed;
    boundary="----=_Part_57_608698271.1689834507143"

------=_Part_57_608698271.1689834507143
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 7bit

Povodny text emailu

 Pridany podpis na koniec
------=_Part_57_608698271.1689834507143--
```

Figure 17: Communication detail - EML source - Adding PlainText to Body

In the following example (listing 56) we will show how we can add e.g. a signature to each sending communication in HTML.

```
...
var htmlVal = ctx.req.getHtmlText();
if(htmlVal != null && htmlVal != ""){
  htmlVal = htmlVal + '<p> </p><p><strong>----------</strong></p><p><strong>FirstName LastName, Company</strong></p>'
  html.set(htmlVal);
}
...
```

Listing 56: ECMA/Javascript for adding text to body - HTML

Result for this request JSON (listing 57):

```
{
    "to": [
        {
            "addressName": "Lukas",
            "emailAddress": "example+domail@dominanz.sk"
        }
    ],
    "htmlBody": "Povodny HTML text emailu\r\n",
    "testmode": false
}
```

Listing 57: Example for POST request for sending email with adding HTML text in script

Result EML:

```
Date: Thu, 20 Jul 2023 06:46:03 +0000 (UTC)
From: doMail-test - TestVM <test[redacted]>
Reply-To: doMail-test - TestVM <test[redacted]>
To: Lukas [redacted] <lukas.[redacted].sk>
Message-ID: <domic.1561.1.1.86241720.59.1689835563224@d[redacted]>
MIME-Version: 1.0
Content-Type: multipart/mixed;
    boundary="----=_Part_59_73219858.1689835563224"

------=_Part_59_73219858.1689835563224
Content-Type: text/html; charset=UTF-8
Content-Transfer-Encoding: 7bit

Povodny HTML text emailu
<p> </p><p><strong>----------</strong></p><p><strong>FirstName LastName, Company</strong></p>
------=_Part_59_73219858.1689835563224--
```

Figure 18: Communication detail - EML source - Adding HtmlText to Body

5.7.5 Track pixel

In the following example (listing 58) we will show how we can add track pixel to html body.

```
...
html();
trackPixel();
trackPixel.add(); //add track prixel to HTML email
...
```

Listing 58: ECMA/Javascript for adding track pixel

Result for this request JSON (listing 59):

```
{
    "to": [
        {
            "addressName": "Lukas",
            "emailAddress": "lukas@example.sk"
        }
    ],
    "htmlBody": "<b>Text emailu</b>",
    "testmode": false
}
```

Listing 59: Example for POST request for sending email with template only with adding track pisel in script

Result EML:

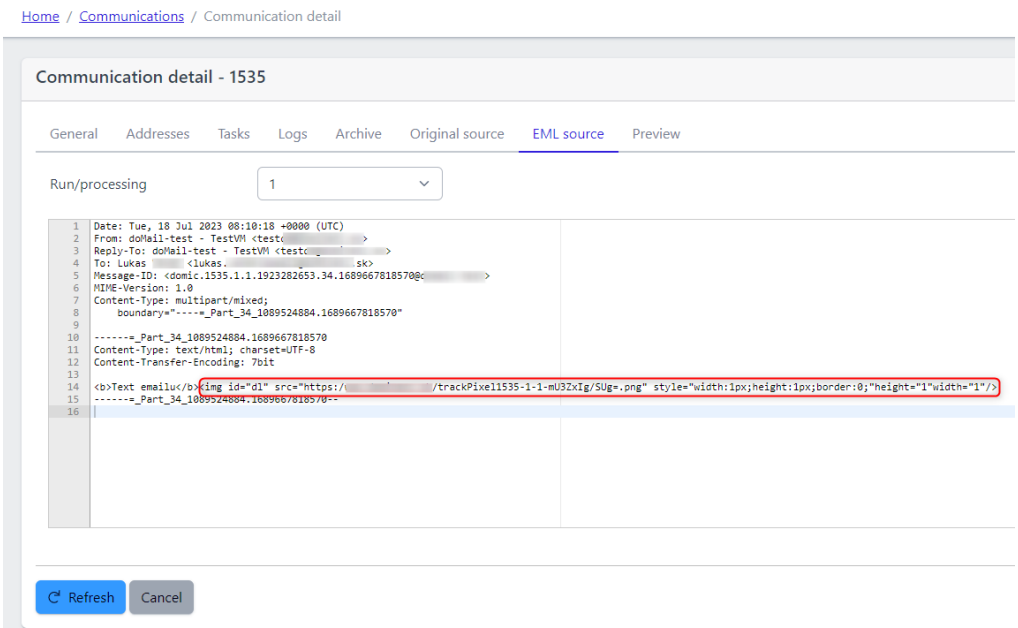

Figure 19: Communication detail - EML source - ExampleEcmaScriptAddingTrackPixel

In case the Track pixel server is running, when the user is reading the email, system doMail will consider the email as delivered and read.

5.7.6 QR codes

5.7.6.1 Add new QR code to end of email

In the following example (listing 60) we will show how we can add generated new QR code to end of HTML email.

```
...
qr.addToEndHtmlPart("www.dominanz.sk");
...
```

Listing 60: ECMA/Javascript for adding new generated QR code

Result for this request JSON (listing 61):

```
{
    "to": [
        {
            "addressName": "Lukas",
            "emailAddress": "lukas@example.sk"
        }
    ],
    "htmlBody": "<b>Text emailu</b>\r\n",
    "testmode": false
}
```

Listing 61: Example for POST request for sending email with generated new QR code

Result (20) EML source:

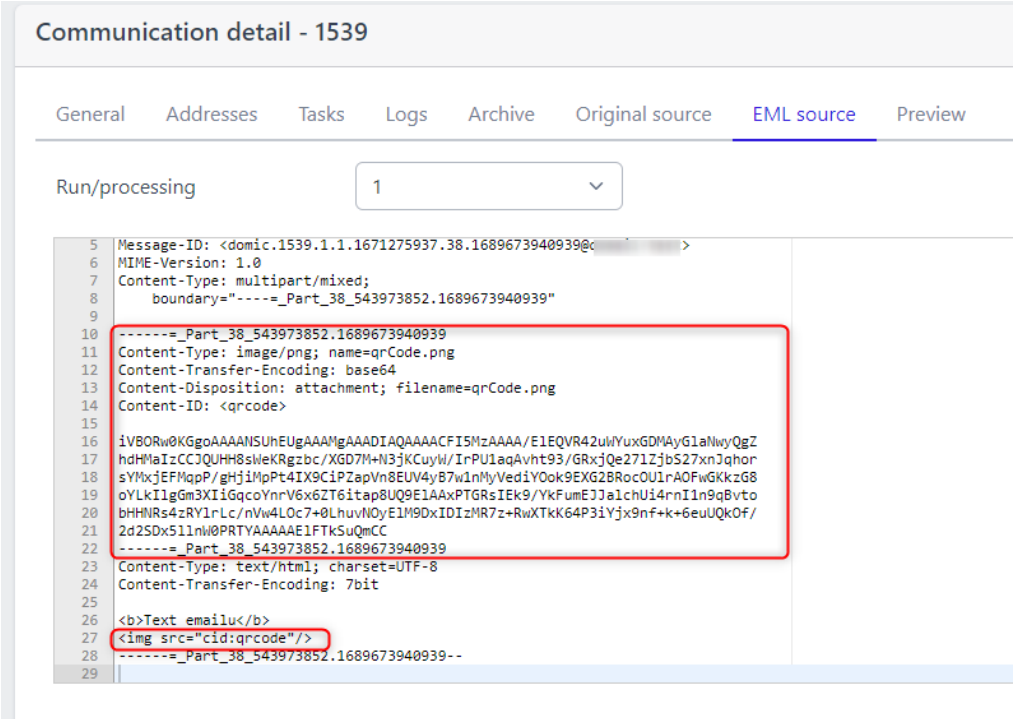

Figure 20: Communication detail - EML source - ExampleEcmaScript QR AddToEndHtmlPart

Result (21) in Email client Outlook:

Figure 21: Outlook - ExampleEcmaScript QR AddToEndHtmlPart

5.7.6.2 Add QR code from iContent

You can add QR codes to the HTML part of an email by setting constants in the scenario to define the size and margin of the QR code. For instance, the size and

margin can be configured as follows:

Selected constants with custom value

Name	Value
qr.code.size	200
qr.code.margin	0

Figure 22: Adding attachment from iContent

The QR code image is then added to the HTML part of the email. The QR code is read from the IContent part of the request. If the request does not contain HTML content, no action is taken. If the HTML does not contain a dedicated section for the QR code, it is automatically appended to the end of the HTML body using the function:

```
...
qr.addToEndHtmlPart();
...
```

Listing 62: ECMA/Javascript for adding attachments from the file system

This ensures that the QR code is always included in the email when needed.

5.7.7 Attachments

In doMail, it is possible to add attachments to emails directly from the system. To configure the system for attachments, set the constant in the scenario for the attachment directory using the following path:

Selected constants with custom value

Name	Value
attachment.dir	$INST_DIR/domail/data/atachments

Figure 23: Adding attachment from system

After setting the directory path, use the `attachment()` function to add the desired attachment to the email. The filename should be replaced with the actual file you wish to attach. Below is an example 63 of how to add an attachment to the email script:

```
...
attachment();
attachment.add("filename");
...
```

Listing 63: ECMA/Javascript for adding attachments from the file system

5.7.7.1 Add attachments from the request

In the following example (listing 64) we will show how we can add attachment from the request.

```
...
attachment();// attachments from request
...
```

Listing 64: ECMA/Javascript for adding attachments from the request

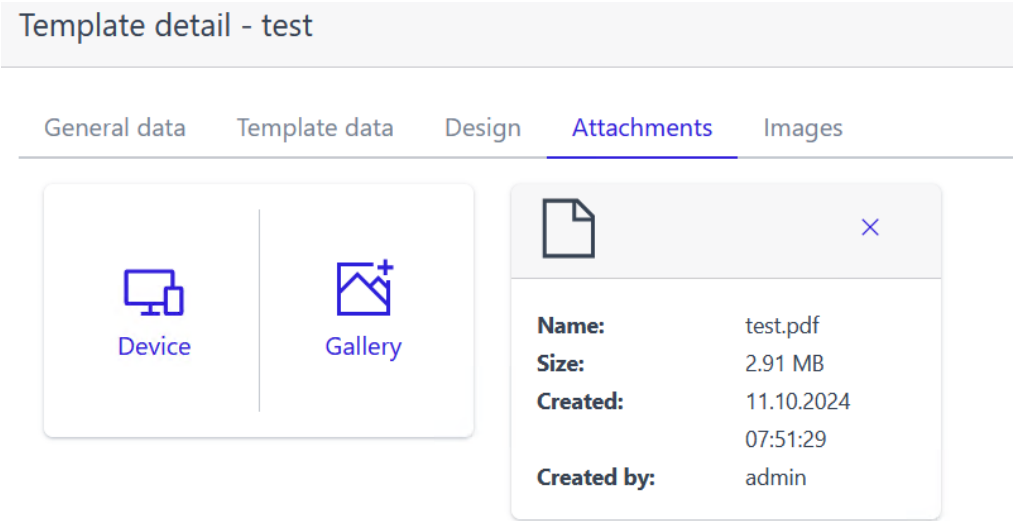

Figure 24: Adding attachment from doMail template

5.7.7.2 Add attachment from Gallery

In the following example (listing 65) we will show how we can add attachment from galery.

```
...
attachment();// attachments from request
attachment.addFromGallery("read.me","MyCustomReadMe");
...
```

Listing 65: ECMA/Javascript for adding attachment from galery

There is result in following figure 25:

```
Date: Wed, 26 Jul 2023 12:26:35 +0000 (UTC)
From: doMail-test - TestVM <[redacted]>
Reply-To: doMail-test - TestVM <[redacted]>
To: Lukas [redacted] <[redacted].sk>
Message-ID: <domic.1623.1.1.535931458.2.1690374394932@[redacted]>
MIME-Version: 1.0
Content-Type: multipart/mixed;
    boundary="----=_Part_2_782191838.1690374394932"

------=_Part_2_782191838.1690374394932
Content-Type: application/octet-stream; name=MyCustomReadMe
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment; filename=MyCustomReadMe
Content-ID: <MyCustomReadMe>

Example of readme file....
------=_Part_2_782191838.1690374394932--
```

Figure 25: Adding attachment from doMail gallery

5.7.8 Certificates, DKIM

5.7.8.1 Domail certificates

In 'Settings -> Certificates -> Domail certificates' you can manage the certificates that are used for signing attachments and emails. On the following figure 26 contains example of our certificate.

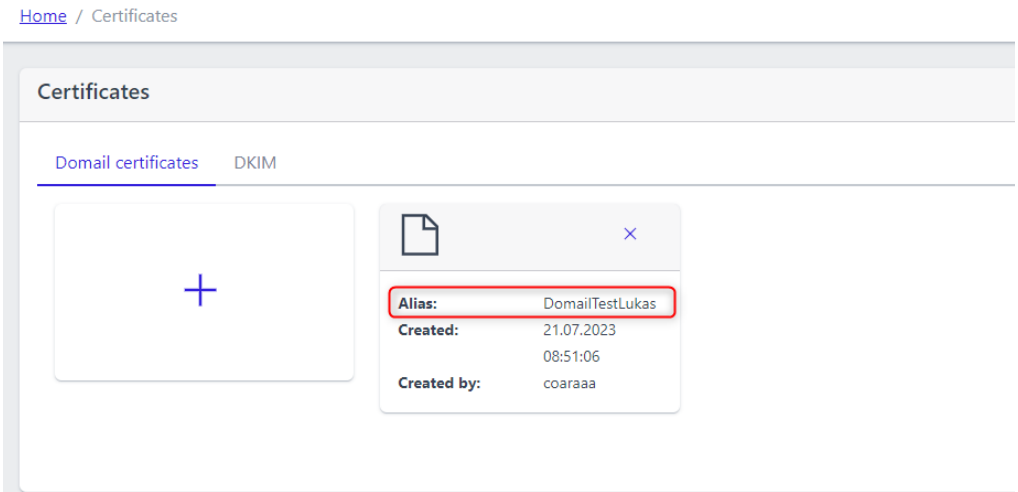

Figure 26: Uploaded certificate for signing attachment

5.7.8.2 DKIM

What DKIM is for? It is a technology for increasing the trustworthiness of emails, which helps to detect spoofed messages. The sent message is signed by the SMTP server with the private key of the sender's domain. This signature is stored in the email header. The receiving server compares this signature with the public key stored in the domain's DNS records. By matching the signature, it is proven that the email actually originated from the sender's domain and was not modified during the transmission of the message.

What are DKIM Selectors? The DKIM selector is specified in the DKIMSignature header and indicates where the public key portion of the DKIM keypair exists in DNS. The receiving server uses the DKIM selector to locate and retrieve the public key to verify that the email message is authentic and unaltered.

How can I find my DKIM Selector? A DKIM selector is specified when the private/public key pair is created when DKIM is set up for the email domain (or email sender), and it can be any arbitrary string of text. The DKIM selector is inserted into the DKIM-Signature email header as an s= tag when the email is sent. The easiest way to discover the selector for your domain is to send an email to yourself. Setting up DKIM (DomainKeys Identified Mail) in Domail involves several essential steps to ensure that your email communications are secure and authenticated. Follow this guide to properly configure DKIM for your domain.

Prerequisites
Before you begin, ensure you have the following:

- Public and Private Key Pair: You need to generate a pair of cryptographic keys (public and private).
- DNS TXT Record: Create a DNS TXT record that includes your public key.

Step-by-Step Setup

- Generate your public and private DKIM keys using a suitable tool or service.
- Add the public key to your DNS records as a TXT record. The DNS record should look something like this:

```
selector._domainkey.example.com IN TXT "v=DKIM1; k=rsa; p=publickey"
```

- Log in to your Domail account.
- Navigate to the Administration Menu.
- Go to the Certificate Screen and select the DKIM Tab.
- Import your PEM file containing both the public and private keys.
- During the import process, you will be prompted to set the following:
  - Alias: This is a freely chosen name to identify the key.
  - Domain: Your domain name (e.g., example.com).
  - Selector: The selector used in your DNS TXT record.

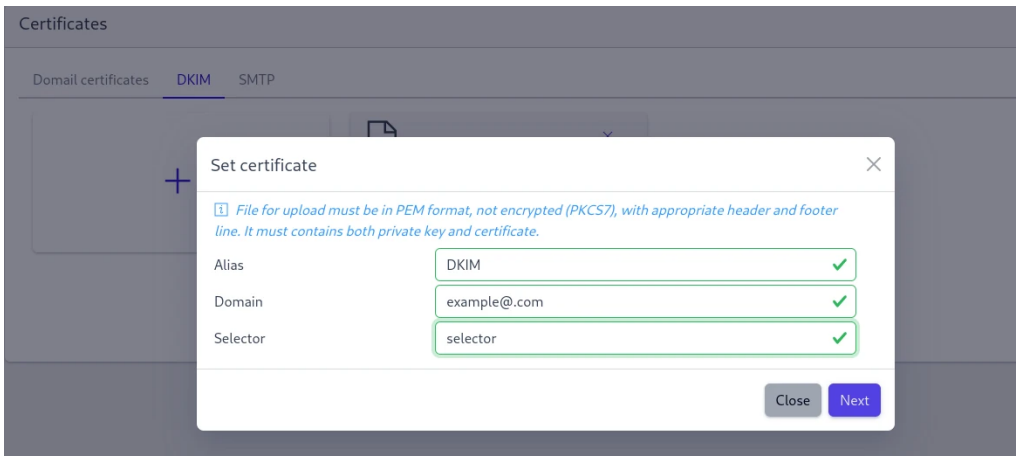

Figure 27: Import DKIM certificate

- After successfully uploading the PEM file, navigate to the Settings Screen.
- Select the DKIM Tab.
- Enable DKIM by configuring the app.dkim.use parameter or manage DKIM verification at the scenario level by setting DKIM constants.

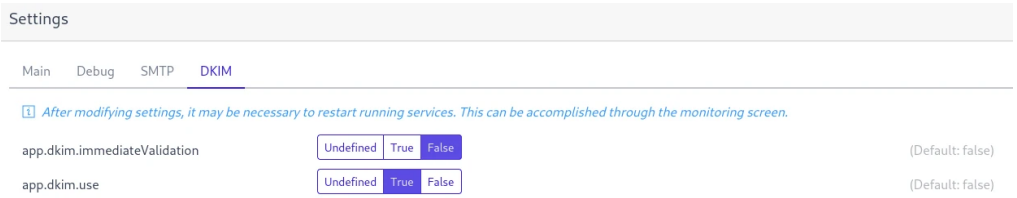

Figure 28: Configuration DKIM Settings

- Modify the existing script to process emails and add the command below:

```
dkim.use("alias");
```

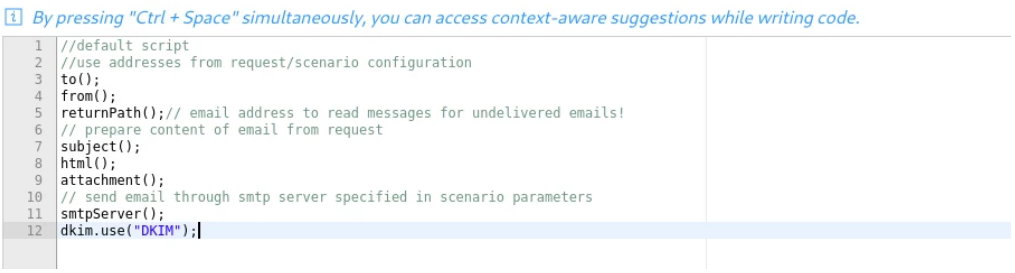

Figure 29: Adding DKIM command to script

- Send a test email to a designated test address.

Figure 30: Example of email with DKIM signature

- Check the email header to verify if it contains a valid DKIM signature.

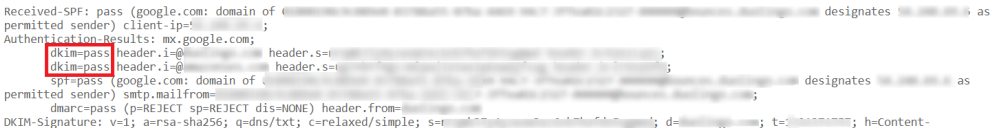

Figure 31: Email header with DKIM signature

5.7.9 Attachments sign

In the following example (listing 66) we will show how we can sign attachments.

```
...
signPdf();//sign attachments from scenario parameters
...
```

Listing 66: ECMA/Javascript for signing attachments

For signing emails it is necessary to upload the certificate and set the custom values constants in the email (32).

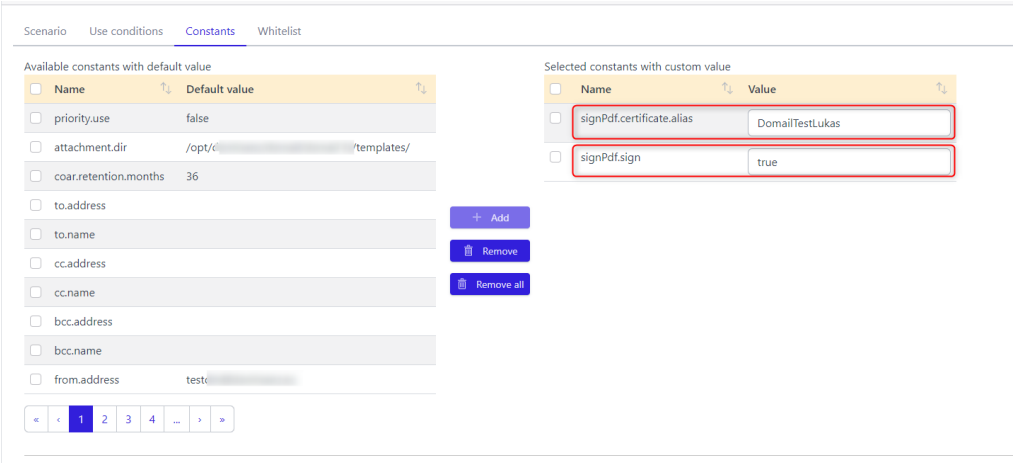

Figure 32: Scenario - set custom values constants for singing attachments

For signing is used the certificate, which was uploaded in the picture 26.

```
Subject: Invoice Attached for Your Review
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="----=_Part_1_2071204135.1728562084860"

------=_Part_1_2071204135.1728562084860
Content-Type: text/html; charset=UTF-8
Content-Transfer-Encoding: 7bit

<html xmlns="http://www.w3.org/1999/xhtml"><head></head><body><p>Dear customer,</p>
<p>I hope this email finds you well.</p>
<p>Please find the invoice attached for your reference. If you have any questions or require further clarification, feel free to reach out to me.</p>
<p>Thank you for your continued business.</p>
<p>Best regards,<br />Your Domail</p></body></html>
------=_Part_1_2071204135.1728562084860
Content-Type: application/octet-stream; name="Invoice 1.pdf"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="Invoice 1.pdf"
Content-ID: <Invoice 1.pdf>


------=_Part_1_2071204135.1728562084860--
```

Figure 33: EML - email with signed attachment

There is a sample of the document 34 signed via domail for your review.

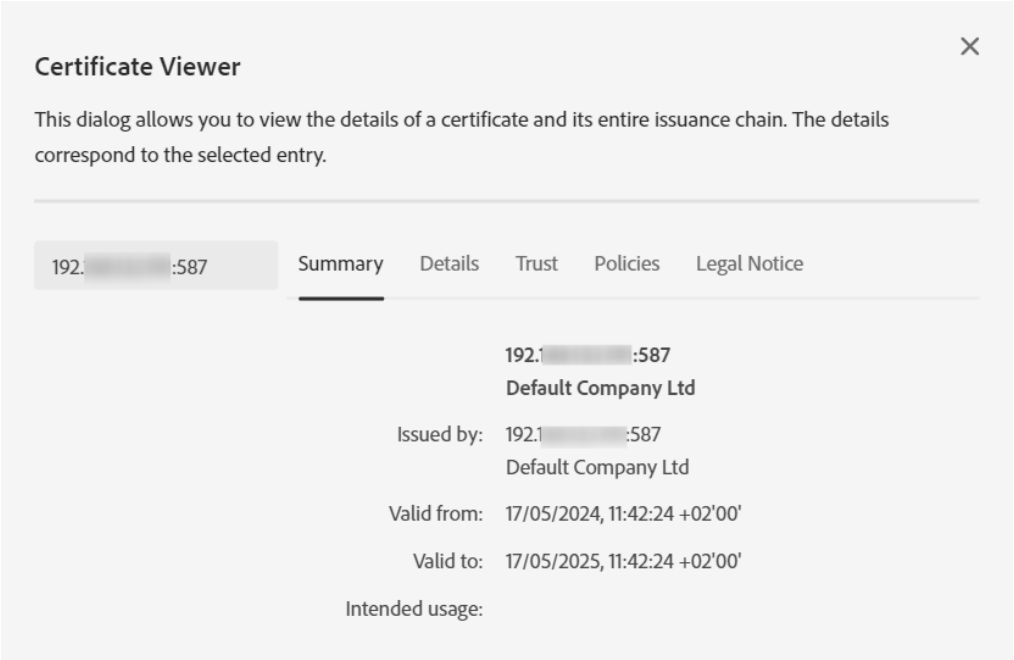

Figure 34: Viewing the signed attachment.

5.7.10 Signing email

In the following example (listing 67) we will show how we can sign email.

```
...
signEmail();
...
```

Listing 67: ECMA/Javascript for signing email

For signing emails it is necessary to upload the certificate and set the custom values constants in the email (35).

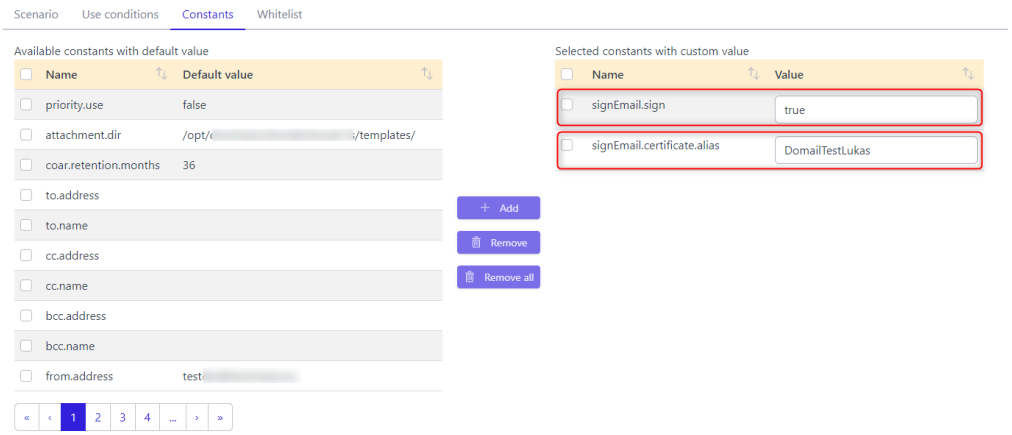

Figure 35: Scenario - set custom values constants for signing email

```
Subject: Signed email
MIME-Version: 1.0
Content-Type: multipart/signed; protocol="application/pkcs7-signature"; micalg=sha-1; boundary="----=_Part_0_784391315.1728566027644"

------=_Part_0_784391315.1728566027644
Date: Thu, 10 Oct 2024 13:13:44 +0000 (UTC)
From: doMail <[redacted]@domail.io>
To: Contact <[redacted]>
Subject: Signed email
Content-Type: multipart/mixed; boundary="----=_Part_2_1489882386.1728566016138"

------=_Part_2_1489882386.1728566016138
Content-Type: text/html; charset=UTF-8
Content-Transfer-Encoding: 7bit

<html xmlns="http://www.w3.org/1999/xhtml"><head></head><body><p>Hello everyone!</p></body></html>
------=_Part_2_1489882386.1728566016138--
------=_Part_0_784391315.1728566027644
Content-Type: application/pkcs7-signature; name=smime.p7s; smime-type=signed-data
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"
Content-Description: S/MIME Cryptographic Signature

------=_Part_0_784391315.1728566027644--
```

Figure 36: EML - signed email with certificate

For signing is used the certificate, which was uploaded in the picture 26.

5.7.11 Context - ctx object

This class contains context objects that scroll between plugins.

- ctx.attachments;
- ctx.fail("Error message");
- ctx.getAttachmentsByExtension(".pdf");
- ctx.getDbEmail();
- ctx.getDbReq();
- ctx.getDbReqRun();
- ctx.id;
- ctx.isFailed();

- ctx.message;
- ctx.message.getMessage();
- ctx.params;
- ctx.req;
- ctx.scenarioParams;
- ctx.scriptConstants;
- ctx.service;
- ctx.service.db;
- ctx.service.fs;

5.7.11.1 attachments

Get List<Attachment<?» object - list of attachments that will be added to email after processing ends.

In the following example (listing 68) we don't send any attachment via request and we want to set the text to PlainText according to the number of attachments.

```
...
if(ctx.attachments.size() > 0){
    text.set("Mam prilohy");
}
else {
    text.set("NEMAM ziadnu prilohu");
}
...
```

Listing 68: ECMA/Javascript context - attachments

There were 0 attachments in the request, result in EML:

```
Date: Mon, 24 Jul 2023 12:58:07 +0000 (UTC)
From: doMail-test - TestVM <test[illegible]>
Reply-To: doMail-test - TestVM <test[illegible]>
To: Lukas [illegible] <lukas.[illegible].sk>
Message-ID: <domic.1599.1.1.1952104131.0.1690203487270@[illegible]>
MIME-Version: 1.0
Content-Type: multipart/mixed;
    boundary="----=_Part_0_2018302915.1690203487282"

------=_Part_0_2018302915.1690203487282
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 7bit

NEMAM ziadnu prilohu
------=_Part_0_2018302915.1690203487282--
```

Figure 37: Context - attachments - result in EML

5.7.11.2 ctx.fail("Error message")

Mark processing as invalid with given description. In the following example (listing 69) we will show how we can failing communication.

```
...
ctx.fail("My custom error from script");
...
```

Listing 69: ECMA/Javascript for failing communication

In the following figure 38 we can see that communication is failing with our cusom error message.

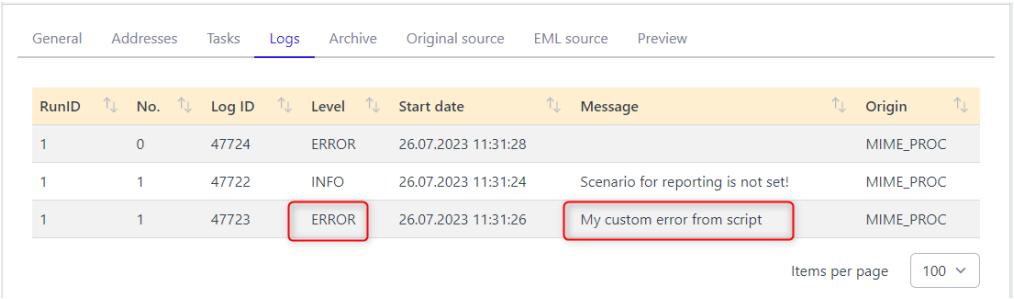

Figure 38: Context - failing communication

5.7.11.3 ctx.getAttachmentsByExtension(".pdf")

Get List<Attachment<?» object - list of attachments that will be added to email after processing ends that have specified extension in their name.

5.7.11.4 ctx.getDbEmail()

Get srv_emails object - database entry of single email

5.7.11.5 ctx.getDbReq()

Get srv_emlreq object - database entry of request

5.7.11.6 ctx.getDbReqRun()

Get srv_procrun object - database entry of run

5.7.11.7 ctx.id

Get IdNumRunid object - id of actual processing composed of id of request, num - index of email and runid - id of run

5.7.11.8 ctx.isFailed()

Returns true if error ocurred during processing, otherwise return false

5.7.11.9 ctx.message

Get MailMessage object for work with generated message

5.7.11.10 ctx.message.getMessage()

Get SMTPMessage object - generated email

5.7.11.11 ctx.params

Get Map<String, Object> object - map of parameters set during processing

5.7.11.12 ctx.req

Get CreateCommunicationRequestI object - request to send email.

In the following example (listing 71) we will show how we can replace a text in plain text body of email. Result for this request JSON (listing 70):

```
{
    "to": [
        {
            "addressName": "Lukas",
            "emailAddress": "lukas@example.sk"
        }
    ],
    "plainTextBody": "Povodny text emailu v systeme $domail$",
    "scenario": "Lukas-SOAP/REST",
    "testmode": false
}
```

Listing 70: Example for POST request for replacing text in body - PLAIN

In the following example (listing 71) we will show how we can replace a text in plaing text.

```
...
var plainText = ctx.req.getPlainText();
if(plainText != null && plainText != ""){
    plainText = plainText.replace('$domail$','doMail');
    text.set(plainText);
}
...
```

Listing 71: ECMA/Javascript for replacing text in body - PLAIN

Result in EML:

```
Date: Wed, 26 Jul 2023 10:13:18 +0000 (UTC)
From: doMail-test - TestVM <test[illegible]>
Reply-To: doMail-test - TestVM <test[illegible]>
To: Lukas [illegible] <lukas [illegible].sk>
Message-ID: <domic.1621.1.1.244258411.0.1690366398293@[illegible]>
MIME-Version: 1.0
Content-Type: multipart/mixed;
    boundary="----=_Part_0_1901670238.1690366398307"

------=_Part_0_1901670238.1690366398307
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 7bit

Povodny text emailu v systeme doMail
------=_Part_0_1901670238.1690366398307--
```

Figure 39: Context - req - replace text in EML

5.7.11.13 ctx.scenarioParams

Get Map<String, String> object - map of parameters set for actual scenario

5.7.11.14 ctx.scriptConstants

Get Map<String, String> object - map of set constants

5.7.11.15 ctx.service

Get services to work with

5.7.11.16 ctx.service.db

Get DbService object - service for work with database

5.7.11.17 ctx.service.fs

Get FsService object - service for work with file system

5.7.11.18 getPlainText

Retrieve the plain text content from the EContentItemTypePlainText type.

5.7.11.19 getScenarioMappingKeyDumpMessage

The method returns a message for logging the scenario (sysId, scenario, ...).

5.7.11.20 getScenarioMappingKeyDump

The method returns the scenario identifier (sysId, scenario, ...) separated by dashes.

5.7.11.21 getEContentValue

Retrieves the value of an EContent item that matches the specified type from the request; returns `null` if no matching item or EContent structure is found.

5.7.11.22 addEContentItem

Adds a new content item to the request, creating the EContent structure if it does not already exist, and sets the content item's type and value.

5.7.11.23 addAttachment

This method enables files to be attached in a format compatible with WSDL standards, using Base64 encoding to ensure proper data transmission.

5.7.11.24 getContentType

Determine the MIME content type of a given file based on its extension.

5.7.11.25 setEContentValue

The method, 'setEContentValue', sets the value of a specific type in the 'eContent' object. If the 'eContent' object or the specified type does not exist, it creates them. The method takes two parameters: 'type' (the type of content) and 'value' (the value to be set).

5.7.11.26 setProcessingCase

The method sets a 'ProcessingCase' object into the header of a request. If the header doesn't exist, it creates one. It uses the 'ProcessingCase' object's scenario, category, and operation to populate the header.

5.7.11.27 hasIContentItem

The method returns true if an iContent with the given name exists in the request and contains a value.

Bibliography

[1] *ECMA-262. ECMAScript® 2023 language specification*. 2023. URL: https://www.ecma-international.org/publications-and-standards/standards/ecma-262/. (accessed: 30.06.2023).

Alphabetical Index